Kurs OMSI
im WiSe 2010/11

Objektorientierte Simulation mit ODEMx

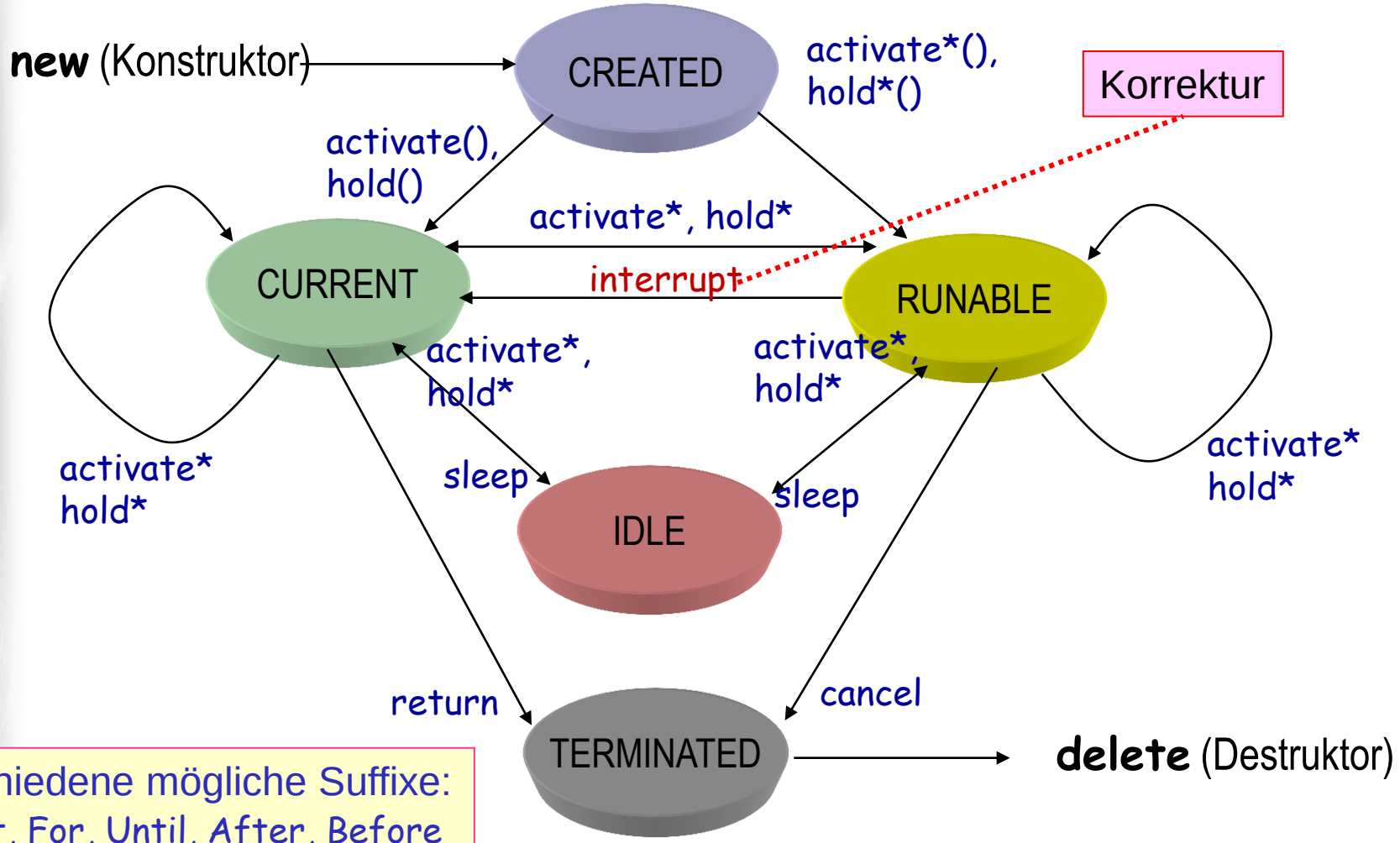
Prof. Dr. Joachim Fischer
Dr. Klaus Ahrens
Dipl.-Inf. Ingmar Eveslage

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

3. Prozess-Scheduling

1. Aufgaben von Klasse Simulation (Wdh.)
2. Process-Listen eines Simulationskontextes
3. Allgemeines Process-Scheduling
4. Weitere Process-Funktionaliät
5. Prozesswarteschlangen: ProcessQueue, Port
6. Spezielles Process-Scheduling (Memory)

Überblick: Zustände und Scheduling-Operationen

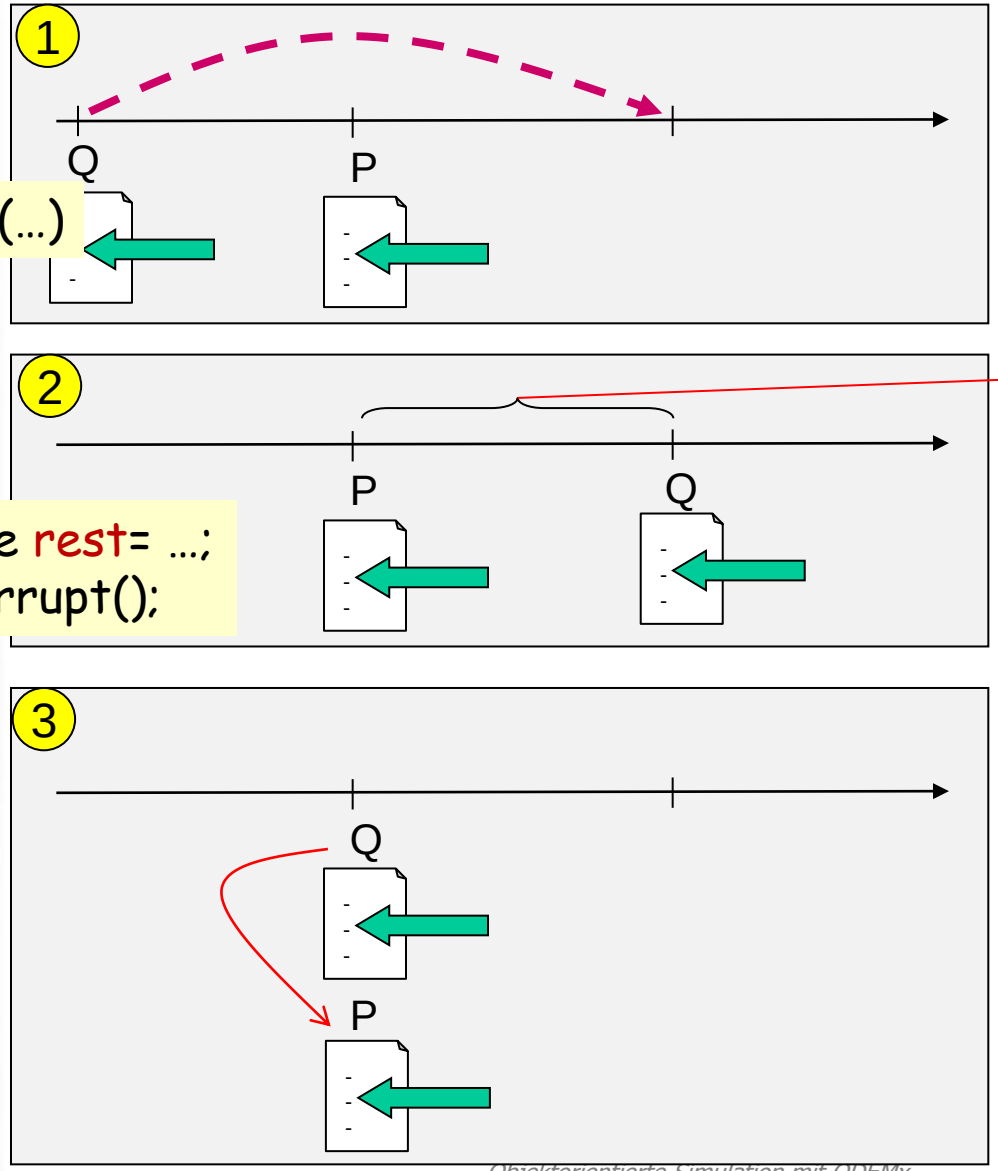


Process: Scheduling-Operationen

Prozessunterbrechungen

```
virtual void interrupt();
        // runable-Prozess wird in seiner hold/activate-Phase unterbrochen
        // wird zum neuen Current-Prozess  (aus Zukunft zurückgeholt), falls kein
        // Prioritätskonflikt
        // und kann mit getInterrupter() die erfolgte Unterbrechung erkennen und
        // selbst behandeln
```

Process: Interrupt-Mechanismus



ODEMx-wait-Funktion: Motivation

typisches Synchronisationsproblem

- Prozesse (z.B. Zustandsmaschinen) setzen ihren Lebenslauf (Zustandsübergänge) nur unter bestimmten Bedingungen fort:
  - in einem von mehreren Eingangspuffern wurde eine Nachricht/Ereignis/Anforderung für den wartenden Prozess hinterlegt
  - ein oder mehrere Zustandsereignisse sind eingetreten
  - ein oder mehrere Zeitereignisse (ausgelöst durch Wecksignale von Uhren) sind eingetreten

- Dabei kann genau ein Prozess oder aber auch mehrere betroffen sein, wobei auch eine selektive Auswahl der Prozesse modellierbar sein sollte

im Lebenslauf eines Prozesses

```
…
result= wait (buffer1, buffer2, timer1, cond1);
switch ( result->getType ) {
   case TIMER: …
   case BUFFER: …
   case CONDITION
   default: …
}
…
```

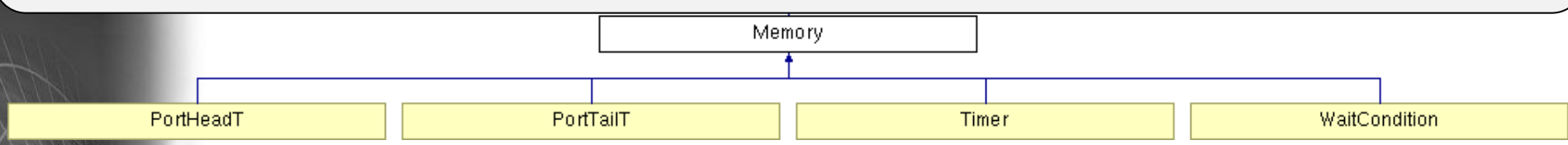
Aufrufer müsste sich jeweils registrieren bei buffer1, buffer2, timer1, cond1

bei diesen Objekten sollten sich weitere Objekte registrieren können

Objekte könnten für Aktivierung der wartenden Prozesse sorgen

Klasse Memory

Basisklasse für
- PortHeadT, PortTailT
- Timer, WaitCondition



Spezialisierungen legen Semantik von available fest

Protected Member-Functionen

```
        Memory (Simulation *sim, MemoryType t, MemoryObserver *mo=0)
virtual ~Memory ()
virtual void alert () //reaktiviert vermerkte Sched-Einträge
virtual Trace * getTrace () const
bool remember (Sched *newObject)
bool forget (Sched *rememberedObject)
void eraseMemory ()  //löscht vermerkte Sched-Einträge
```

} Verwaltung von memoryList

Statusinformation eines Memo-Objektes

```
virtual bool available ()=0 // testet die Bereitschaft  des Memo-Objektes.
MemoType getMemoryType () // liefert den Typ.
```

Protected Attribute

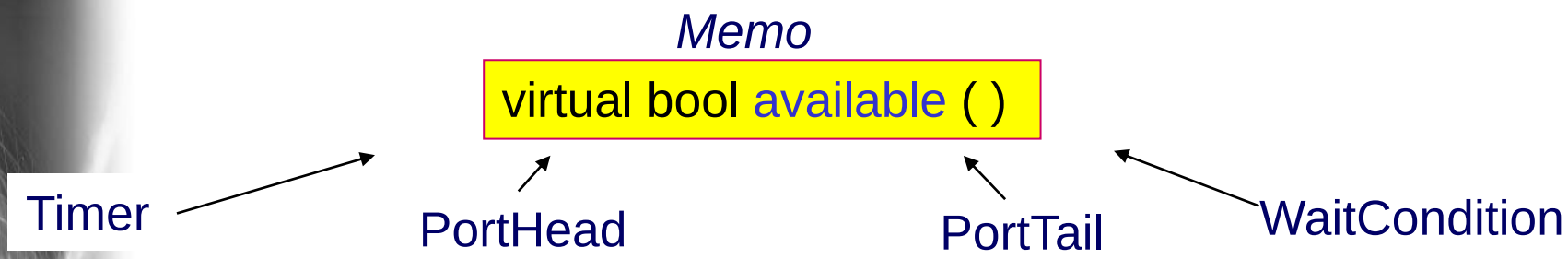
```
std::list< Sched * > memoryList //Liste vermerkter Sched-Objekte (Zeiger)
```

```
enum MemoryType {
    TIMER,
    PORTHEAD,
    PORTTAIL,
    CONDITION
};
```

Memory-Funktionalität

bei Alarmierung: alert()-Aufruf

- Aktivierung der blockierten Sched-Objekte zum aktuellen Zeitpunkt
- abermalige Ausführung der spezifischen available-Funktion
- Erneute Blockierung oder Ausführungsfortsetzung mit Verlassen der Memory-Liste



Timer	PortHead	PortTail	WaitCondition
virtual bool available ( )	virtual bool available ( )	virtual bool available ( )	virtual bool available ( )
Returns: true if this timer is not scheduled	**Returns:** true if the buffer is not empty	**Returns:** true if the buffer is not full	**Returns:** condition-result
Timeout-Aktion beendet: blockierte (Prozesse warteten auf Zeitsignal (-verfügbarkeit))	blockierte Empfangs-prozesse warteten auf Nachrichtenverfügbarkeit	blockierte Sende-prozesse warteten auf Pufferverfügbarkeit	blockierte Prozesse warteten auf Bedingungserfüllung

Wait: Benutzung von Timern und Ports

- Process-Member-Funktion wait

```
Memo* wait (Memo* m0,
            Memo* m1= 0, Memo* m2= 0, Memo* m3= 0, Memo* m4= 0, Memo* m5= 0 )
```

```
Memo* wait (MemoVector* memvec  )
```

liefert eines der Memo-Objekte zurück, sobald dieses „Verfügbarkeit“ liefert;
bis dahin bleibt der Aufrufer blockiert

- Anwendungsbeispiele

```
PortHead *p1, *p2, *p3, *p;
```

```
...
p= wait (p1,p2, p3);
```

Aufrufer-Prozess wartet(blockiert) bis in einem der Buffer eine Nachricht abgelegt wird

```
Memo *m;
PortHead *ph;
PortTail *pt;
Timer t;
```

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

Aufrufer-Prozess wartet(blockiert) bis in einem der Puffer ph eine Nachricht abgelegt wird oder In einem Puffer pt Platz geworden ist oder ein Timeout anliegt

PortHead, PortTail als Klassen und Templates

Nachrichtentyp
Parameter der Typschablonen

Konstruktoren

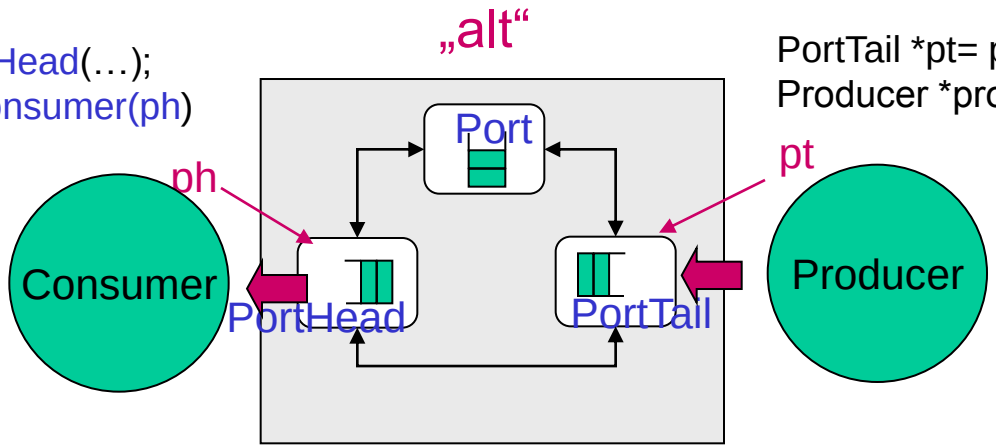
```
PortHeadT (Simulation * sim,
           Label portName,
           PortMode m = WAITING_MODE,
           unsigned int max = 10000,
           PortHeadObserverT< ElementType > * pho = 0 )

PortTailT (Simulation * sim,
           Label portName,
           PortMode m = WAITING_MODE,
           unsigned int max = 10000,
           PortHeadObserverT< ElementType > * pho = 0 )
```

PortMode-Aufzählungstyp

ERROR_MODE	Fehler, falls voll/leer
WAITING_MODE	Prozesswechsel, falls voll/leer
ZERO_MODE	Leeranweisung, falls voll/leer (0 als Return-Wert)

Weitere Port-Funktionalität

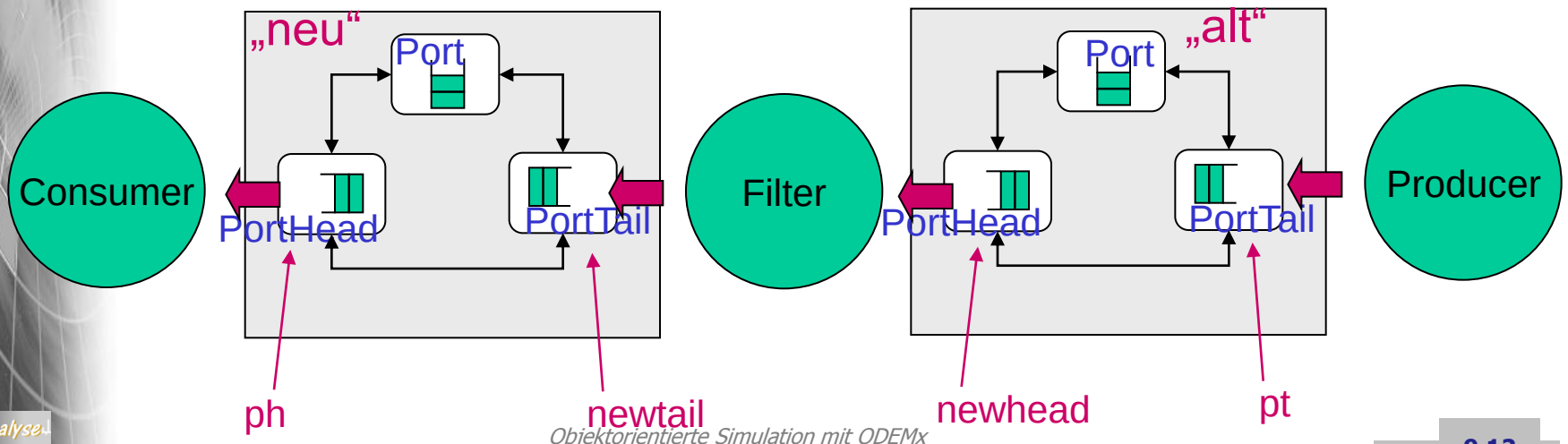
PortHead *ph= new PortHead(…);
Consumer *con= new Consumer(ph)



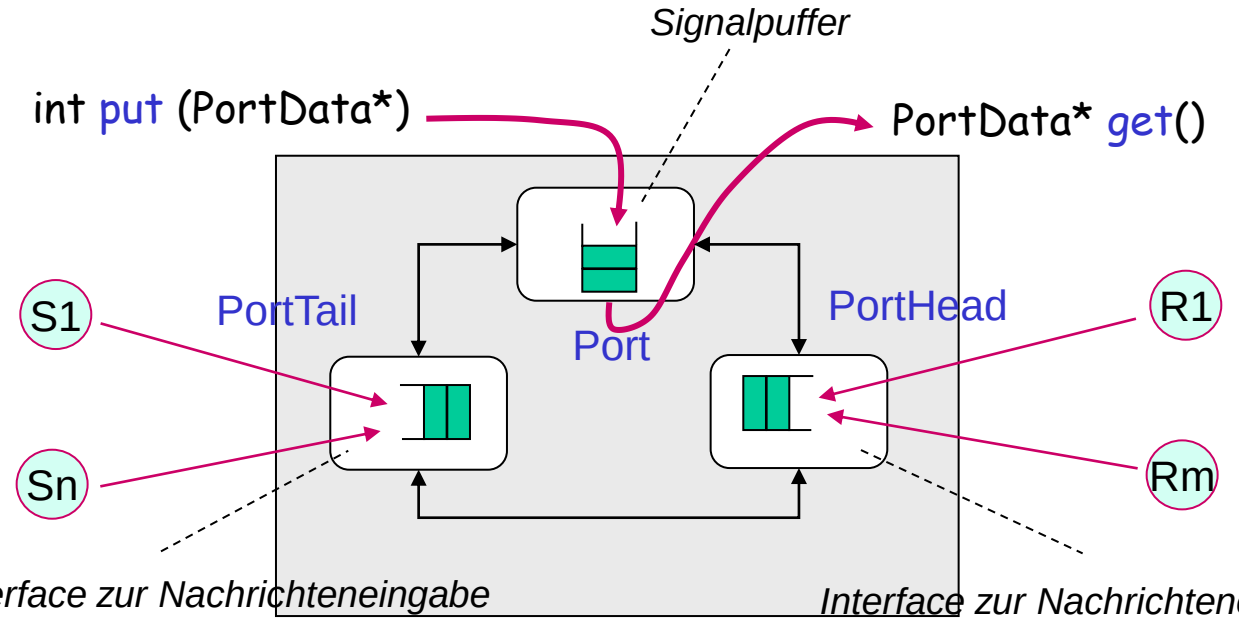
PortTail *pt= ph.tail()
Producer *pro= new Producer(…, pt)

PortHead *newhead= ph->cut();
PortTail *newtail= ph->tail();

Filter *filter= new Filter(…, newhead, newtail)



Prozesskommunikation zum asynchronen Signalaustausch



- Ensemble von OdemX-Klassen
  PortHead, PortTail und Port
  bildet einen asynchronen Nachrichtenpuffer
- Nachrichten sind Ableitungen von PortData

ODEMx-Templates
PortHeadT, PortTailT, PortT

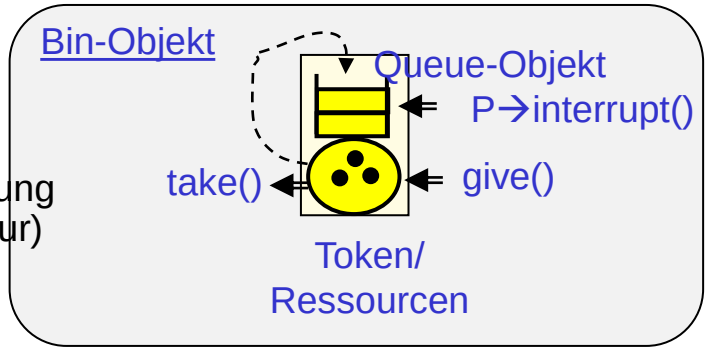
aktueller Template-Parameter

Auto-Klasse als PortData-Typ

4. ODEMx-Modul Synchronisation:
Bin, Res

- Verwaltung geteilt genutzter Ressourcen mittels Bin und Res
- Die Klasse Bin
- Behandlung von Unterbrechungen
- Die Klasse Res
- Allgemeine Prozesslokalisierung (nicht Bin/Res-spezifisch)
- Ein Beispiel: Autofährbetrieb (Einsatz von Bin und Res)

Klassen Bin und Res



Gemeinsamkeiten

- verwalten **Ressourcen** und deren geteilte Nutzung
  Ressourcen sind abstrakt (Token ohne Struktur)

- bieten Funktionen zur
  - **Anforderung** und
  - **Freigabe** von Ressourcen

- ein Prozess wird in seiner Anforderung **blockiert**
  und im Queue-Objekt **vermerkt**,
  falls die Ressourcen in geforderter Anzahl z.Z. **nicht** zur Verfügung stehen

- ein blockierter Prozess wird **fortgesetzt**,
  sobald die von ihm benötigte Anzahl von Token zur Verfügung steht
  bei Entnahme der Token

Unterschiede

- Bin kann **beliebig viele** Token (unabhängig von der initialen Ausstattung) aufnehmen, erlaubt dynamische Vernichtung und Generierung von Token
- Res **beschränkt** die maximal verfügbare Token-Menge,
  geht i.d.R. von einer unveränderlichen Token-Menge je Res-Objekt aus

4. ODEMx-Modul Synchronisation:
Bin, Res

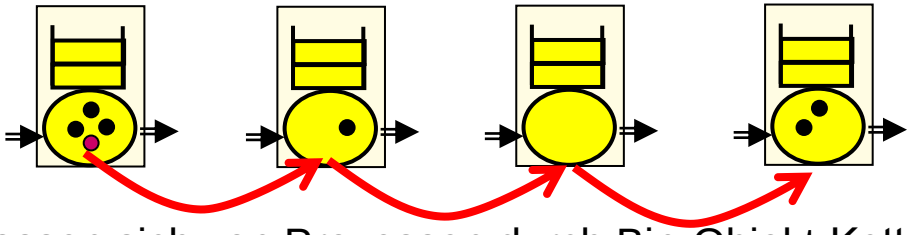
- Verwaltung geteilt genutzter Ressourcen mittels Bin und Res
- Die Klasse Bin
- Behandlung von Unterbrechungen
- Die Klasse Res
- Allgemeine Prozesslokalisierung (nicht Bin/Res-spezifisch)
- Ein Beispiel: Autofährbetrieb (Einsatz von Bin und Res)

Klasse Bin: Konzept

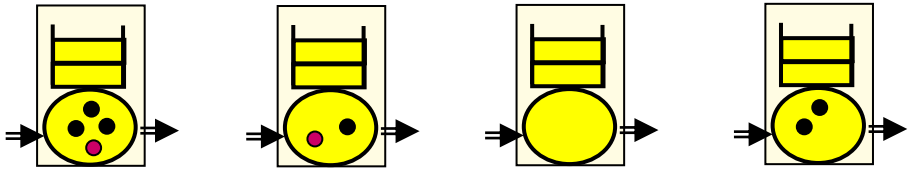
- besitzt zur Verwaltung blockierter Prozesse (privates) Queue-Objekt: Queue* takeWait
- per Konstruktorparameter wird initiale Tokenzahl (>=0) vermittelt (Defaultwert 0)
- Member-Funktion int take(n), n>0
  - Rufer (**Nehmer-Rolle**) wartet evtl. mit Null-Zeit als „Durchläufer“
  - i.d.R. wartet der Rufer solange (d.h. ohne Unterbrechung), bis n Token verfügbar sind
  - aber: Warteaktion ist prinzipiell durch nebenläufigen Prozess unterbrechbar
    - Rückgabewert zeigt erfolgte Unterbrechung an: 0 (sonst n)
- Member-Funktion give(n), n>0
  - Rufer (in **Geber-Rolle**) veranlasst (zeitlose) Eingabe oder Rückgabe von Token, verbunden mit Aktivierung evtl. wartender Nehmer-Prozesse
  - Token müssen nicht zwingend demselben Bin-Objekt zurück gegeben werden

Bin-Anwendungen

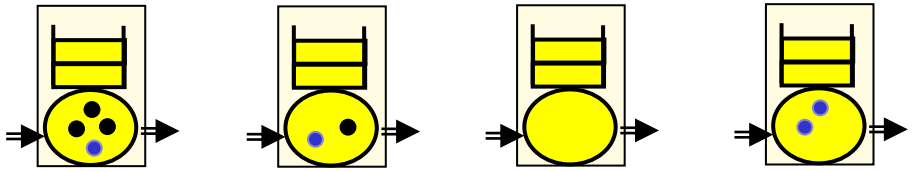
Token haben jedoch **keine** Identität



Token lassen sich von Prozessen durch Bin-Objekt-Ketten ‚bewegen'



Token lassen sich von Prozessen in Bin-Objekt-Ketten erzeugen
(ohne dass sie zuvor irgendwo entnommen wurden)



Token lassen sich von Prozessen in Bin-Objekt-Ketten vernichten
(ohne dass sie danach irgendwo abgelegt werden)

Klasse Bin: Schnittstelle und Implementation (1)

```
class Bin : public Observable<BinObserver>,
        public DefLabeledObject, public StatisticObject,
        public virtual TraceProducer, public virtual ReportProducer {

    // Implementation
    private:
        Simulation* env;
        unsigned int tokenNumber;
        unsigned int initToken;
        Accum tokenStatistics;
        unsigned int users;
        unsigned int providers;
        double sumWaitTime;

        // process management
        Queue takeWait;
```

Accum wurde auch
zur Warteschlangenstatistik eingesetzt
s. letzte Vorlesung: Queue

Klasse Bin: Schnittstelle und Implementation (2)

prinzipiell für **alle** Kontextbezogene ODEMx-Objekte

Public Member Functions

Bin (Label l, unsigned int startTokenNumber, BinObserver *o=0) Construction for DefaultSimulation.

Bin (Simulation *s, Label l, unsigned int startTokenNumber, BinObserver*o=0) Construction for user-defined Simulation.

~Bin () Destruction.

const std::list< Process * > & getWaitingProcessList () const Get list of blocked processes.

virtual Trace * getTrace () const Get pointer to trace.

virtual void report (Report *r) Generate report.

Token management

unsigned int take (unsigned int n) Take n token.

void give (unsigned int n) Give n token.

unsigned int getTokenNumber () const Number of tokens available.

Statistics

virtual void reset () reset statistics

unsigned int getUsers () const Number of takes.

unsigned int getProviders () const Number of gives.

unsigned int getInitial () const Initial number of token.

unsigned int getMin () const Min number of token.

unsigned int getMax () const Max number of token.

double getAVFreeToken () const Average free token.

double getAVWaitTime () const Average waiting time.

Implementierung von Bin::take

```
unsigned int Bin::take(unsigned int n) {

    takeWait. inSort(getCurrentProcess());

    if (n>tokenNumber or
            takeWait. getTop() != getCurrentProcess()) {
            … // Statisik
            while (n>tokenNumber or takeWait.getTop()!=getCurrentProcess()) {
                    getCurrentProcess()->sleep();
                    // Unterbrechung des rufenden Prozesses
                    // …
                    // Fortsetzung nur bei Aktivierung durch anderen Prozess
                    if (getCurrentProcess()->isInterrupted()) {
                            takeWait. remove(getCurrentProcess());
                            return 0;
                    }
            }
            …// Statistik
    }
    tokenNumber -= n;
    takeWait.remove(getCurrentProcess());
    awakeFirst(&takeWait); //Aktivierung des nächsten blockierten Prozesses vom Bin-Objekt
    return n;
}
```

Implementierung von Bin::give

```
void Bin::give(unsigned int n) {
    ...
    tokenNumber += n;
    ...
    awakeFirst(&takeWait);
   //Aktivierung des nächsten blockierten Prozesses vom Bin-Objekt
}
```