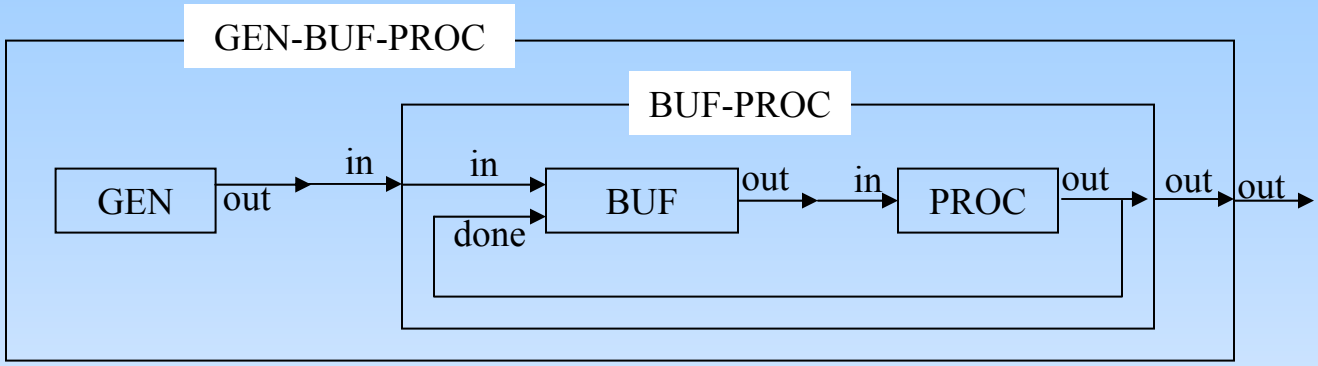
DEVS Modeling Example: GBP

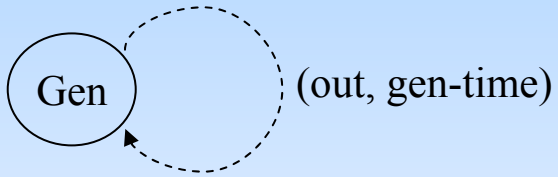


- GEN-BUF-PROC: overall coupled model
  - Informal description
    - Imagine a bank to deposit money
    - Customers arrive, wait at BUF(line) and get service at PROC(teller)
  - GEN: generation of customers
  - BUF+PROC: coupled model
    - BUF: waiting line for PROC to be free
    - PROC: provide service for each customer

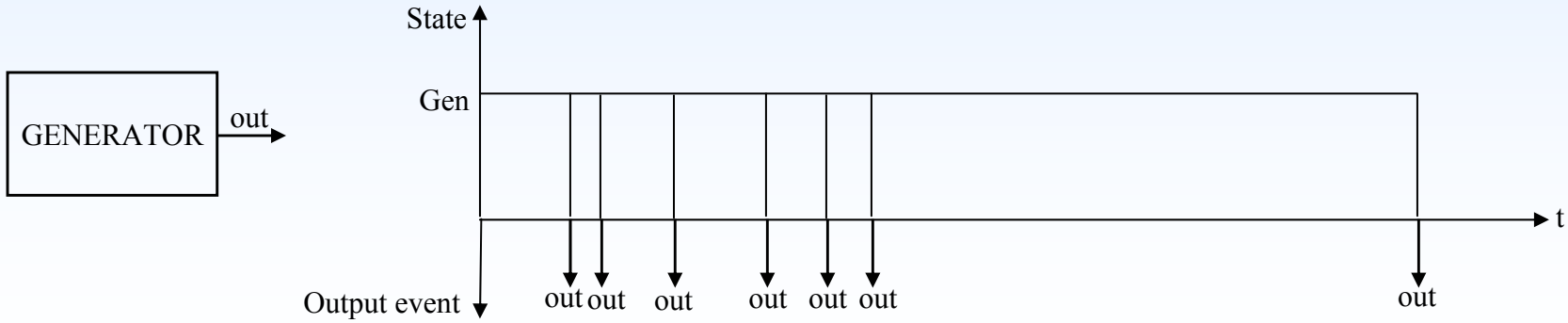
★ Note: Generality of GBP Model
  - PROC: CPU, Teller, Router, Traffic Controller

Atomic Model - GEN

- GEN = < S, X, Y, δ_{int} , δ_{ext} , λ, ta >
  - X, δ_{ext} : not defined
  - Y = {out }
  - S = { Gen }



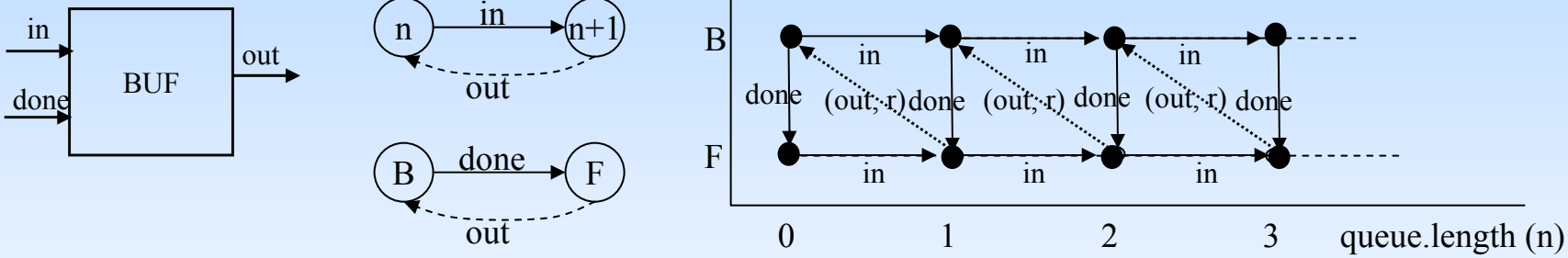
  - ta(Gen) = gen-time (random number)
  - δ_{int}((Gen, ta(Gen)) = (Gen, 0)
  - λ ((Gen, ta(Gen)) = out



Atomic Model - BUF

❖BUF = < S, X, Y, δ_{int} , δ_{ext} , λ, ta >

- X = { in, done }
- Y = {out }
- S = { (n, proc.status) | n ∈[0 … maxlength], proc.status ∈{Busy, Free} }



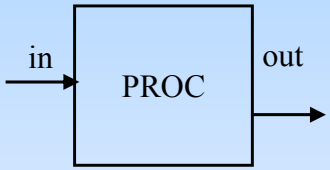
- ta((n≠0 , F)) = ta-old – e if continuation of existing schedule;

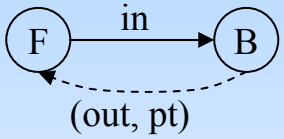
  = sending-time (random number) otherwise

  ta((n=0. -) or (-, B)) = ∞ (waiting for an input)
- δ_{int} ((n≠0 , F) , ta(s)) = ((n-1, B), 0)
- δ_{ext}((n, -), e, x) = ((n+1, -), 0); δ_{ext}((n, B), ∞, done) = ((n, F), 0);
- λ ((n≠0 , F) , ta((n≠0 , F))) = out

Atomic Model – PROC

- PROC = < S, X, Y, δ_{int} , δ_{ext} , λ, ta >
  - X = { in }
  - Y = {out }
  - S = { Busy, Free}





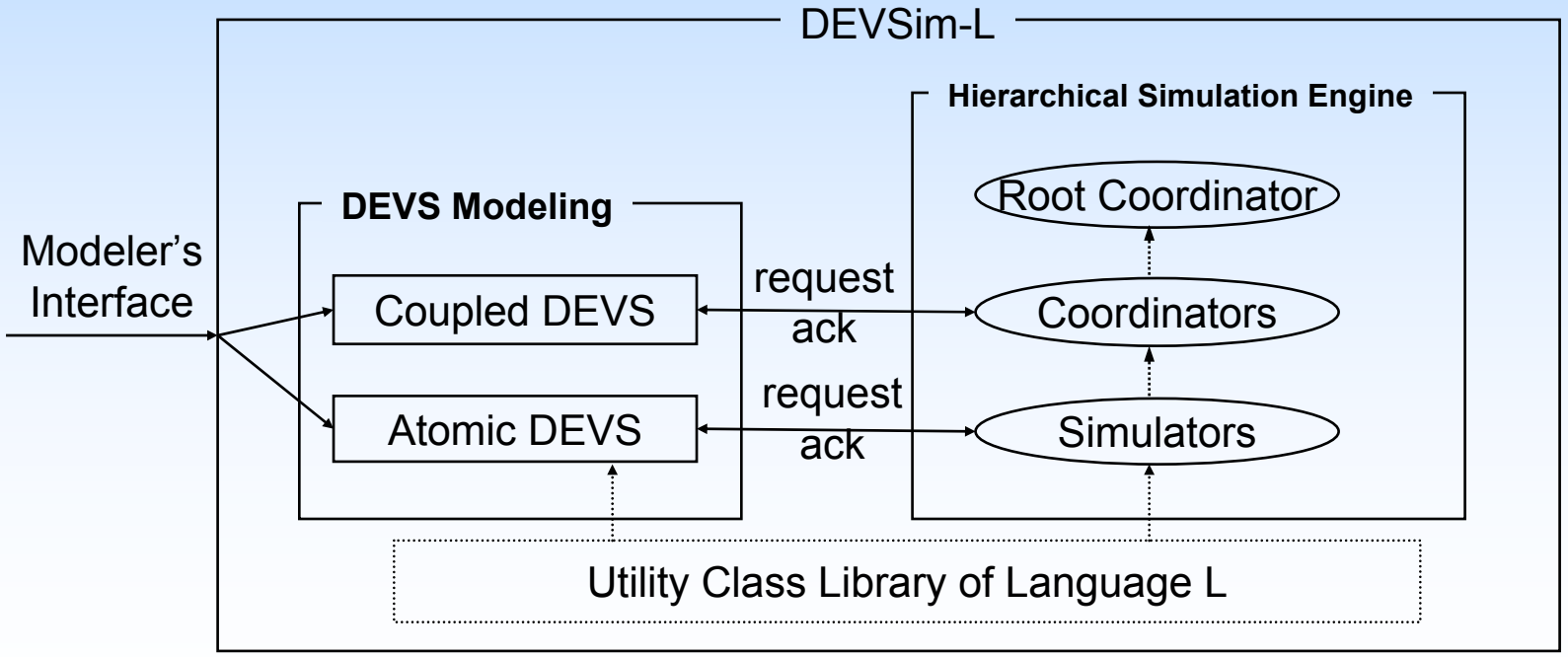
- ta(B) = processing-time (random number) ; ta(F) = ∞
- δ_{int} ((B , ta(B)) = (F, 0)
- δ_{ext}((F, e), x) = (B, 0);
- λ ((B, ta(B)) = out

Coupled Models – BP and GBP

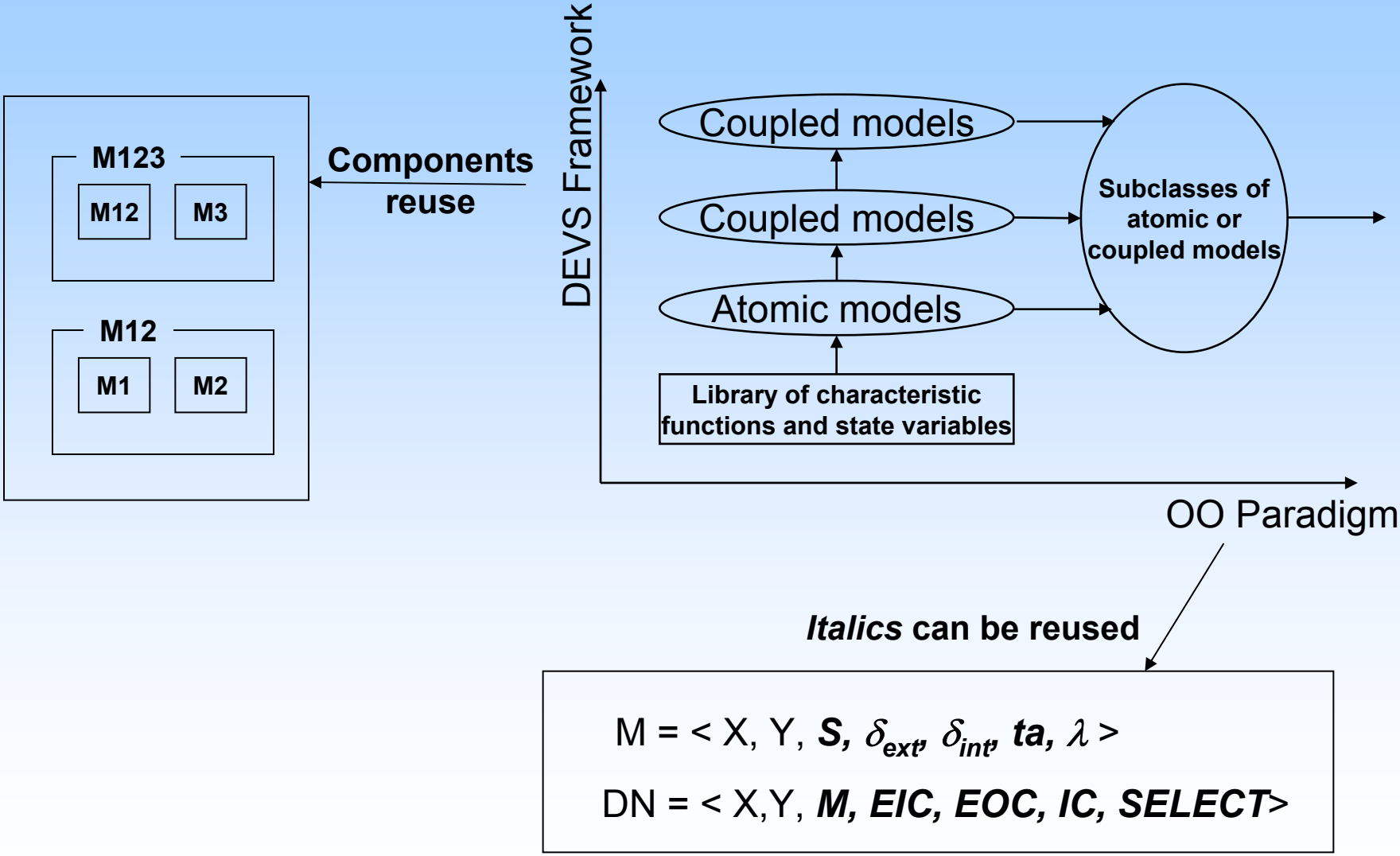
- GBP (GEN+BUF+PROC)
  - BP = < X, Y, {Mi}, EIC, EOC, IC, sel >
    - X = {in }
    - Y = {out}
    - {Mi} = {BUF, PRO }
    - EIC = { (BP.in, BUF.in) }
    - EOC = { (PROC.out, BP.out) }
    - IC = { (BUF.out, PROC.in) (PROC.out, BUF.done) }
    - sel({BUF, PROC}) = PROC
  - GBP = < X, Y, {Mi}, EIC, EOC, IC, sel >
    - X = { }
    - Y = {out}
    - {Mi} = { GEN, BP }
    - EIC = { }
    - EOC = { (BP.out, GBP.out) }
    - IC = { (GEN.out, BP.in) }
    - sel({GEN, BP}) = GEN

DEVS M&S Environment with Language L: DEVSim-L

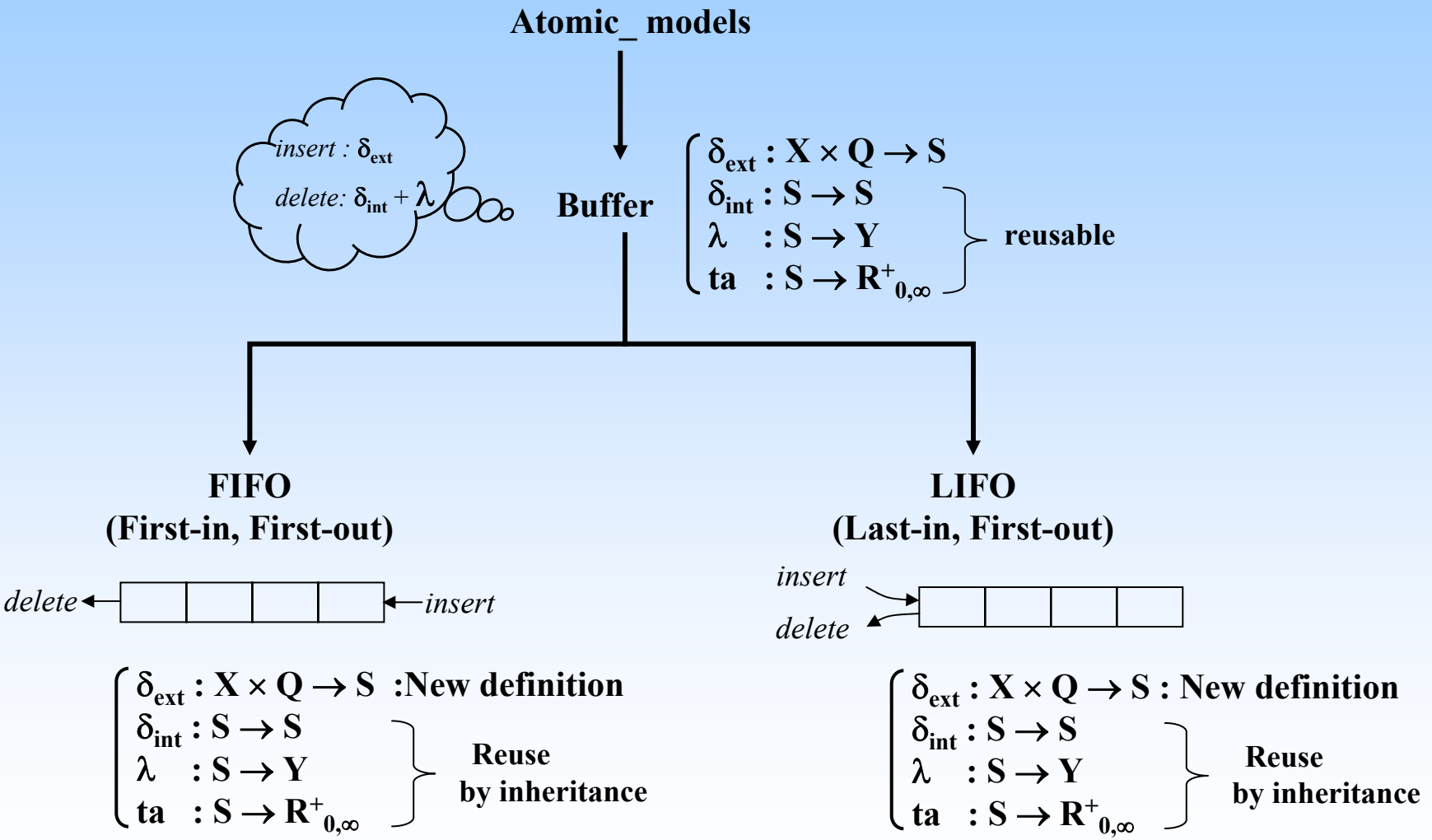
- Discrete Event System Modeling/Simulation Tool
- Realization of the DEVS formalism in C++ / Java
- DEVS Formalism and Object-oriented Paradigm
- Separating Models from Simulation Engine
- DEVSim++ (C++); DEVSimJava (Java); DEVSimHLA (HLA)



Two Dimensional Reuse in DEVSim-L (L: OOP Lang.)



Reuse Through Inheritance: Subclass of Atomic DEVS



DEVSim++ Code : Buffer, FIFO, LIFO


```
class  Buffer : public CAtomic
{
        ...
};

virtual Buffer::ExtTransFn(const CMessage
&message)
{
        ...
}

bool Buffer::IntTransFn()
{
        ...
}

bool Buffer::OutputFn(CMessage &message)
{
        ...
}

TimeType Buffer::TimeAdvanceFn()
{
        ...
}
```

Inheritance

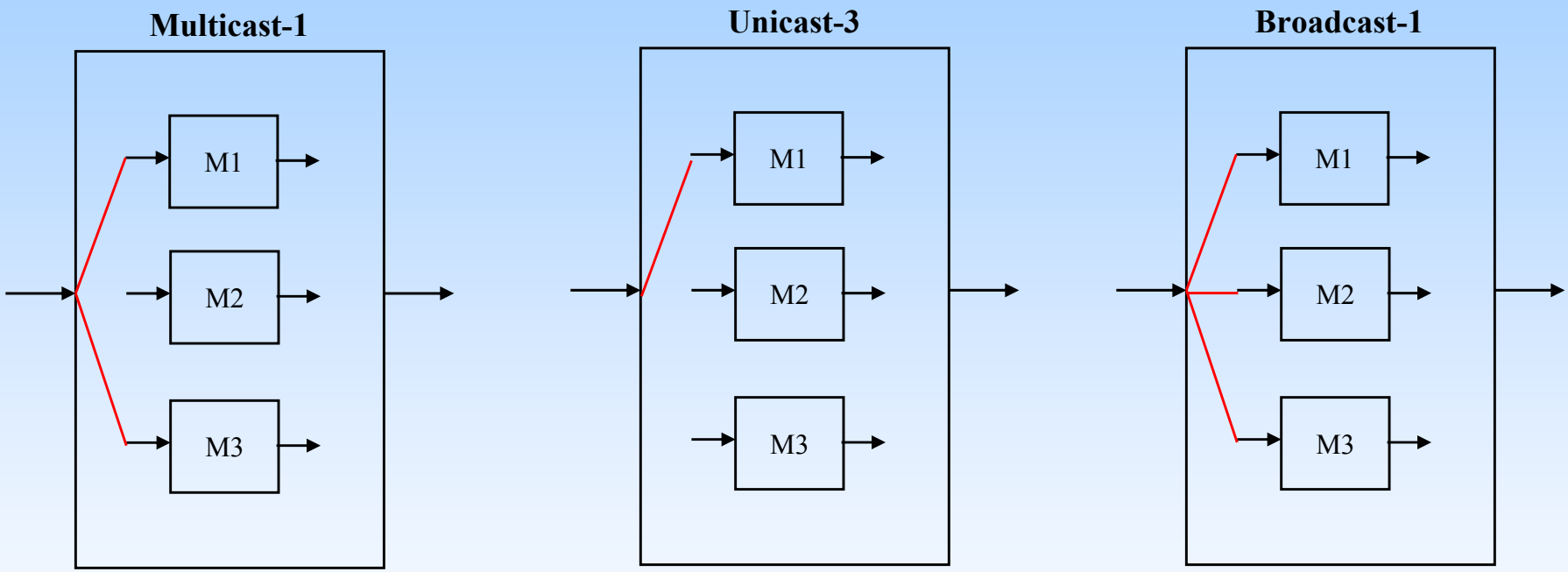
```
class FIFO : public Buffer
{
        ...
};

bool FIFO::ExtTransFn(const Cmessage &message)
{
        ...
}
```

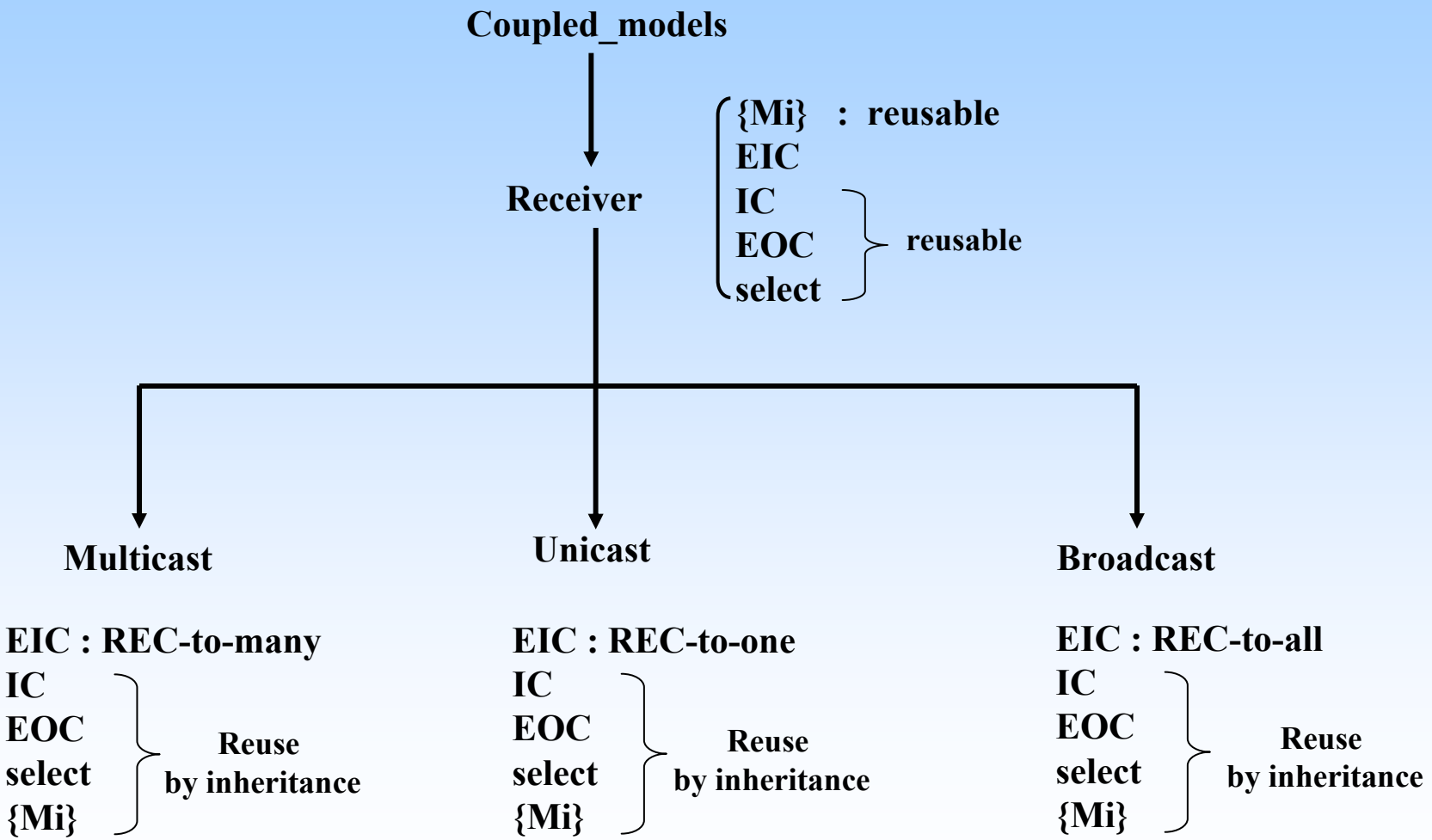
```
class LIFO : public Buffer
{
        ...
};

bool LIFO::ExtTransFn(const CMessage &message)
{
        ...
}
```

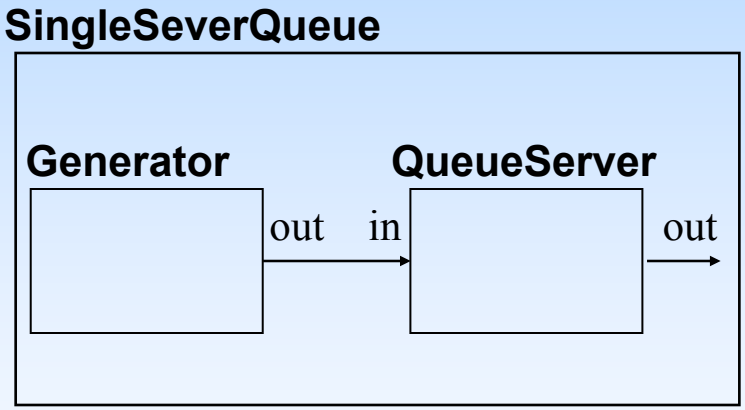
Reuse Through Inheritance: Subclass of Coupled DEVS



Example of Subclass of Coupled Models



DEVSim++ Code For SingleServerQueue

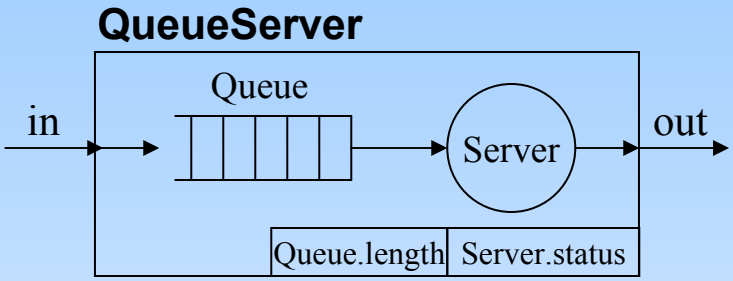


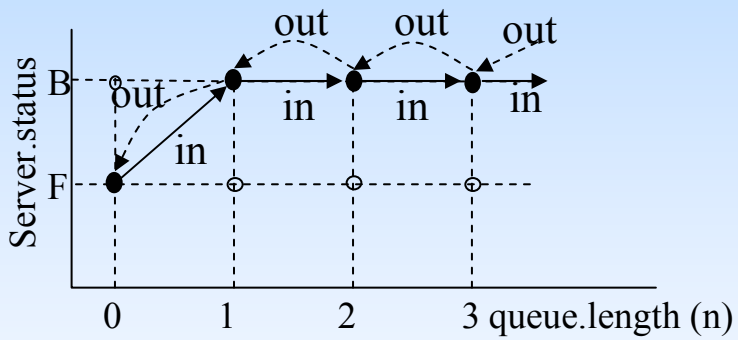

```
class SingleSeverQueue : public CCoupled
{
public:
      SingleSverQueue();
      ~SingleServerQueue();
};

SingleServerQueue::SingleServerQueue()
{
      CModel *Gen, *QueSer;
      Gen = new Generator;
      QueSer = new QueueServer;
      AddComponent(2, Gen, QueSer);
      SetPriority(2, Gen, QueSer);
      AddCoupling(Gen, ”out", QueSer, ”in");
}

.
.
.
```

DEVSim++ Code For QueueServer





X = {in}
Y = {out}
S = {(length,status)| length ∈I, status ∈{B, F}}
$\delta_{ext}((n\neq0,-),in) = (n+1,-)$
$\delta_{ext}((0,F),in) = (1,B)$
$\delta_{int}((n\neq1,B)) = (n-1,B)$
$\delta_{int}((n=1,B)) = (0,F)$
$\lambda((n\neq0,-)) = out$
$ta((n\neq0,-)) = Service\_time$

```
bool QueueServer::OutputFn(CMessage &message)
{
    if(!IsState(length, 0))
        message.SetPortValue(out, NULL);
    return true;
}

TimeType QueueServer::TimeAdvanceFn()
{
    if(IsState(length, 0))
        return Service_Time;
    else return Infinity;
}
```