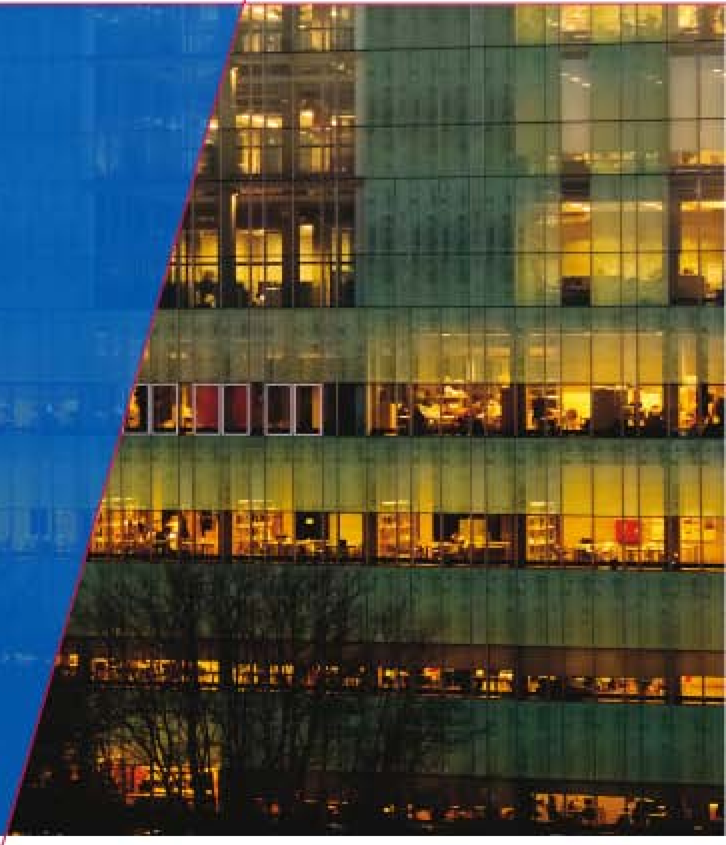
# Hierarchical Petri nets

prof.dr.ir. Wil van der Aalst



Technische Universiteit
Eindhoven
University of Technology

Where innovation starts

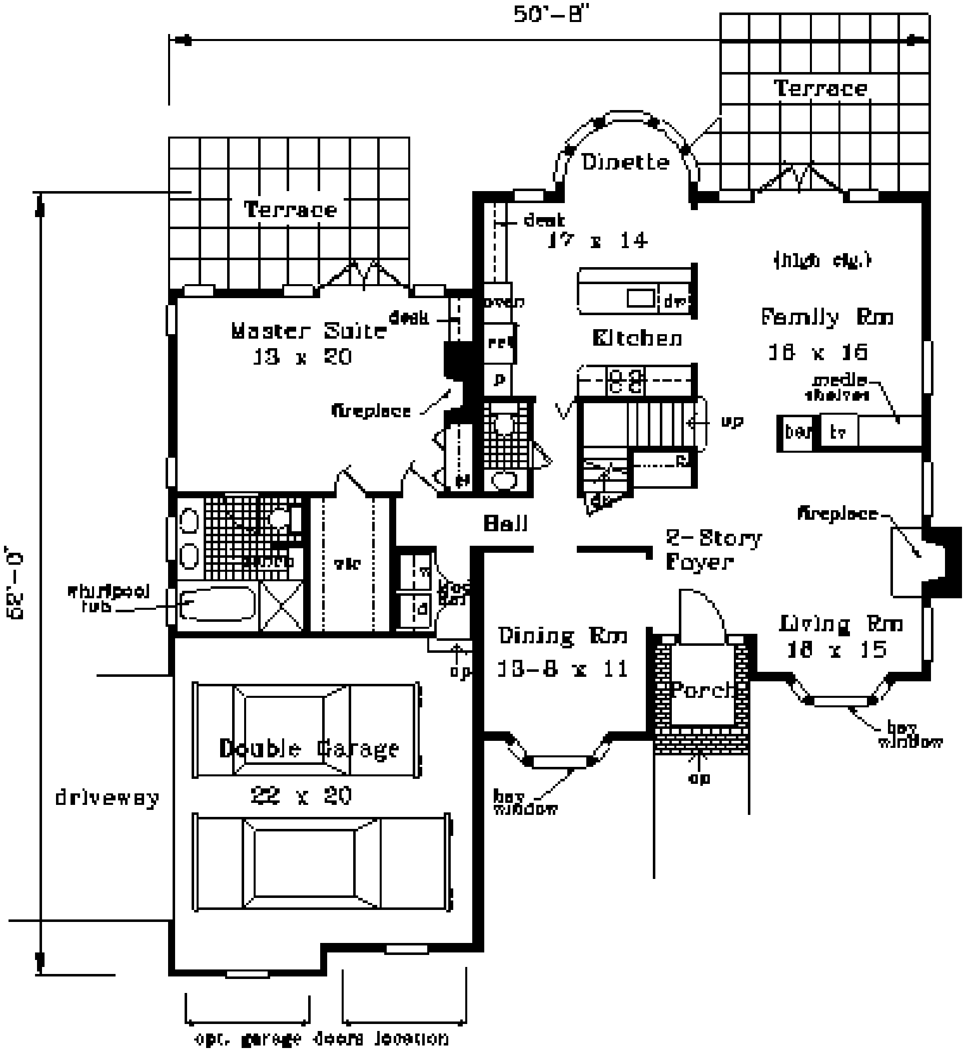
# Extensions of the classical Petri net

- ***Color*** **(i.e., tokens have values, places have color sets, arc have inscriptions, transitions have guards, etc.) was introduced in Chapter 5 and formalized in Chapter 6.**
- ***Time*** **(i.e., tokens have timestamps, color set may be timed, arcs have delays, etc.) was introduced in Chapter 5 and formalized in Chapter 6.**
- **Hierarchy is introduced and formalized in Chapter 7.**
- **A colored, timed, hierarchical Petri net is called a high-level Petri net.**
- **Hierarchical CPNs (=CPN + hierarchy) provide a concrete language for high-level Petri nets.**

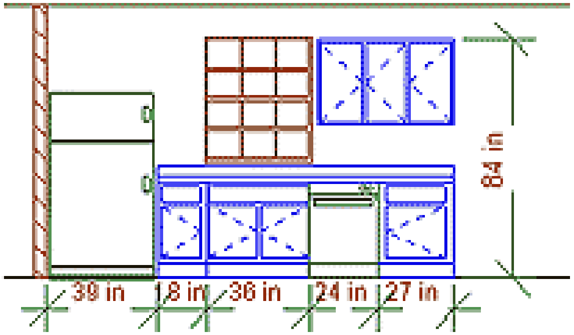
# The need for hierarchy: Compare with construction drawings of a house

- **The top-level: The house as a whole**

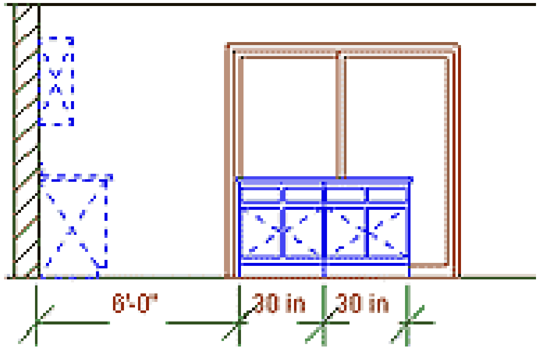
# The second level: The first floor of the house.



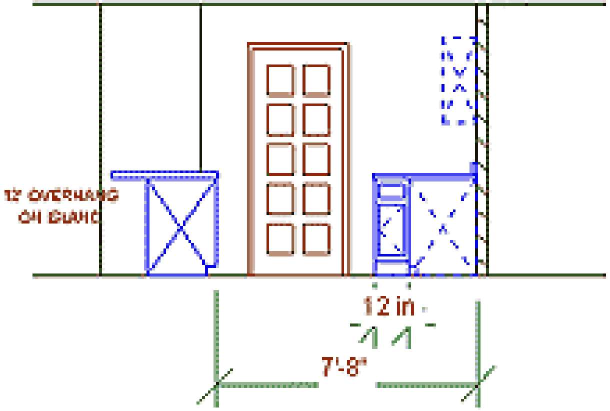
# The third level: The kitchen



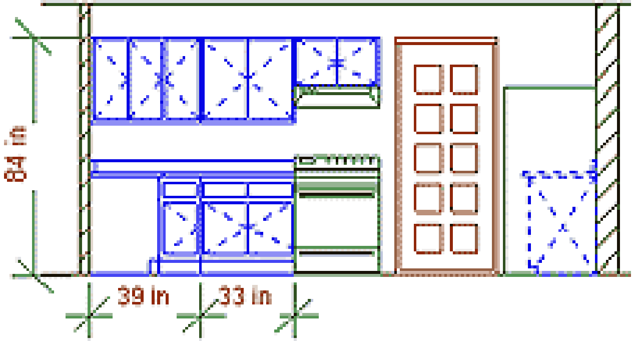
KITCHEN ELEVATION - NORTH



KITCHEN ELEVATION - EAST

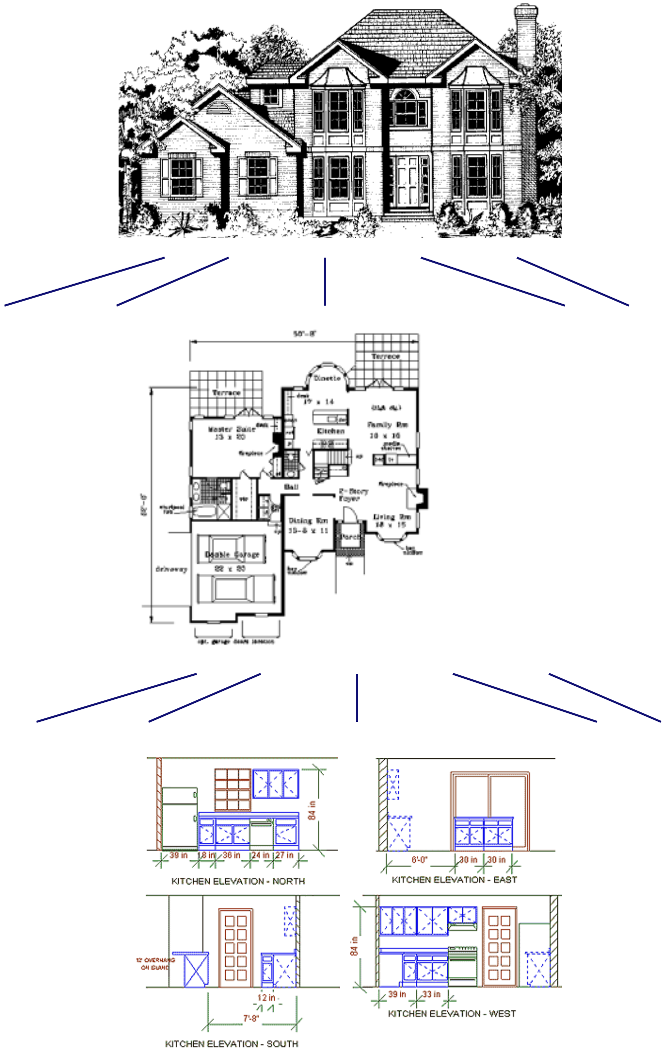


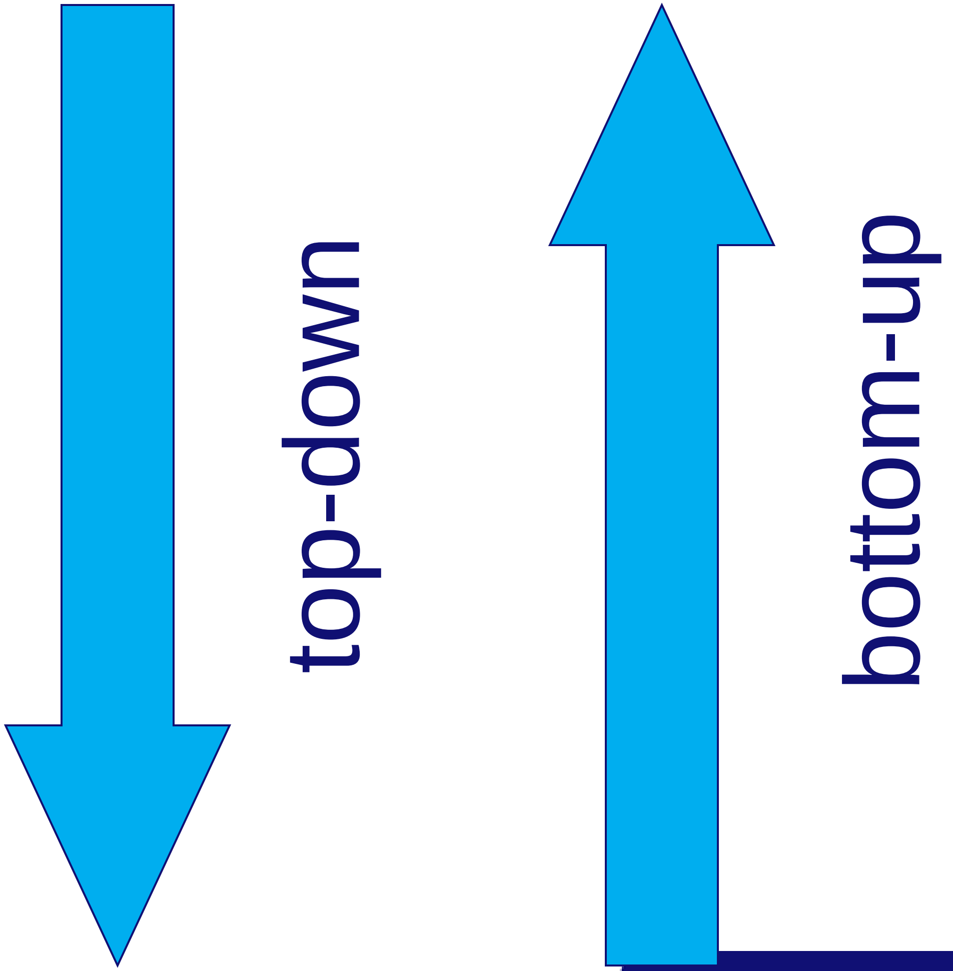
KITCHEN ELEVATION - SOUTH



KITCHEN ELEVATION - WEST

# Top-down versus bottom-up




top-down

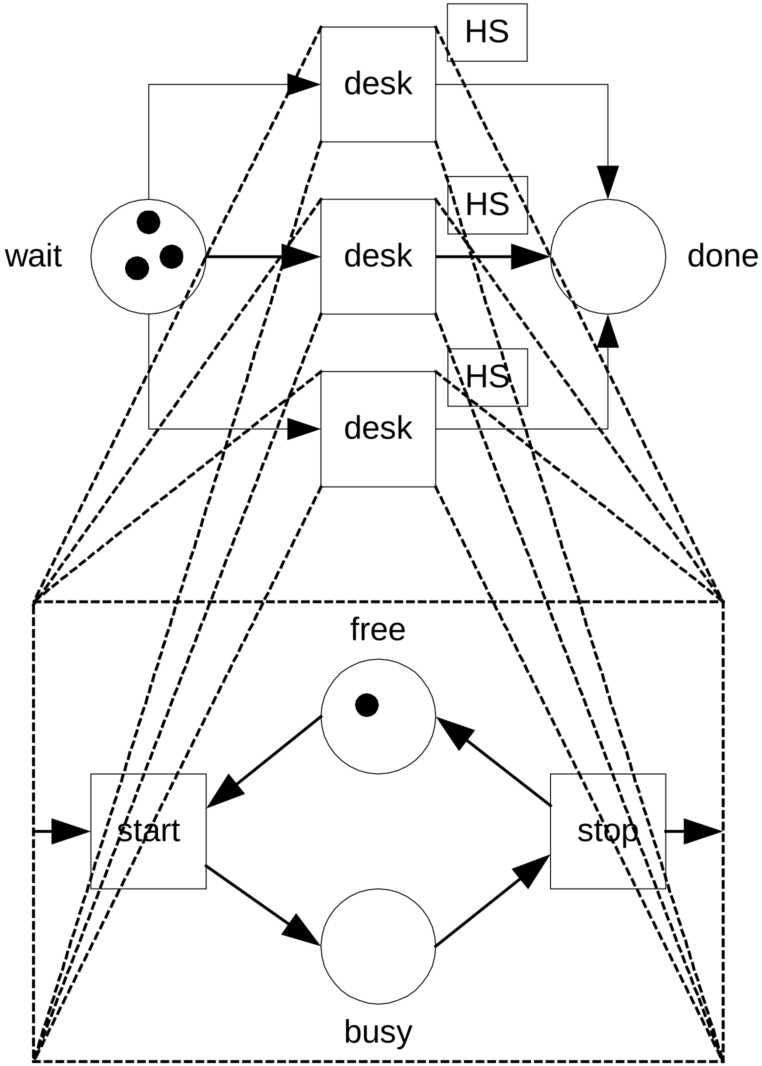
bottom-up

# Recall

- Three good reasons for making a process model:
  - gain insight
    for a better understanding of the system
  - analysis
    validation and verification
  - specification
    a blue print for construction
- Like the construction drawing of an architect!
- However, despite the addition of color and time, a hierarchy concept is still missing thus far!

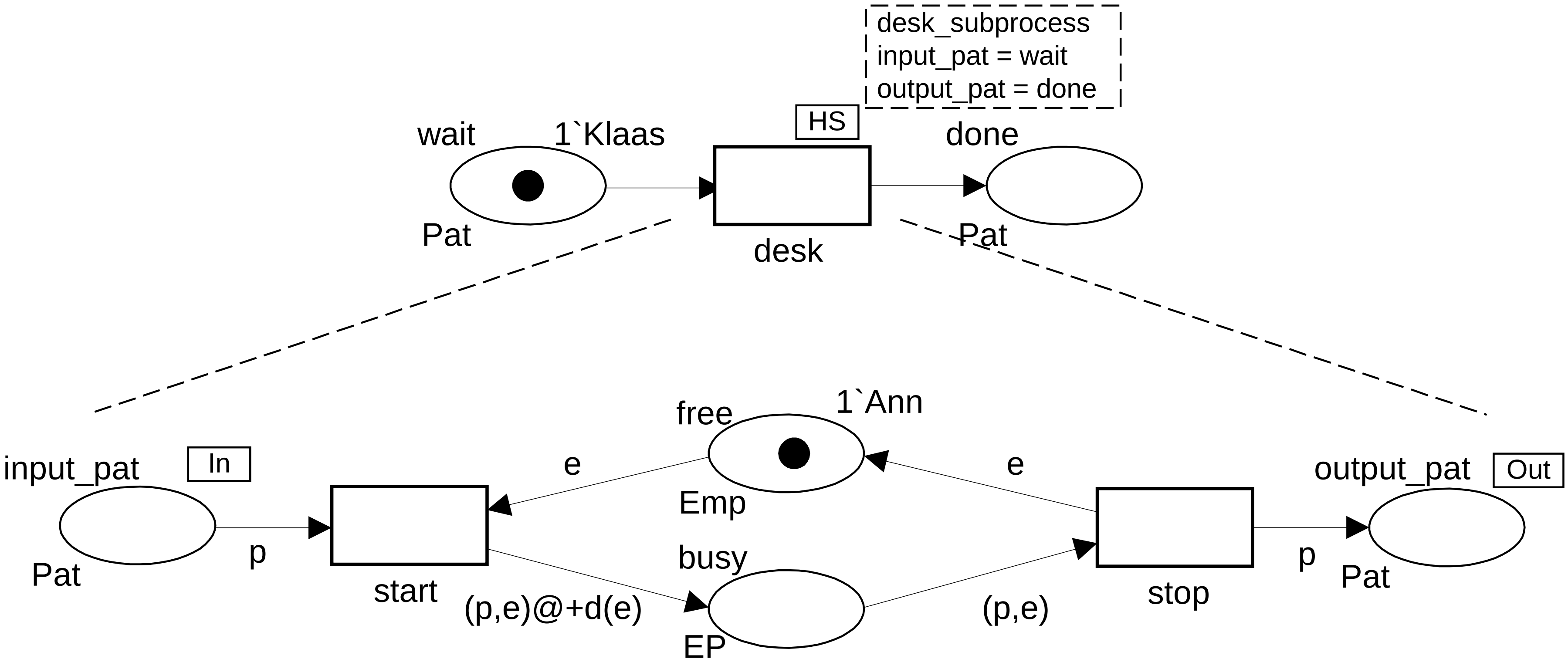
# Basic idea

- **Transitions correspond to subsystems/subprocesses**
- **Divide an conquer**
- **Reuse**

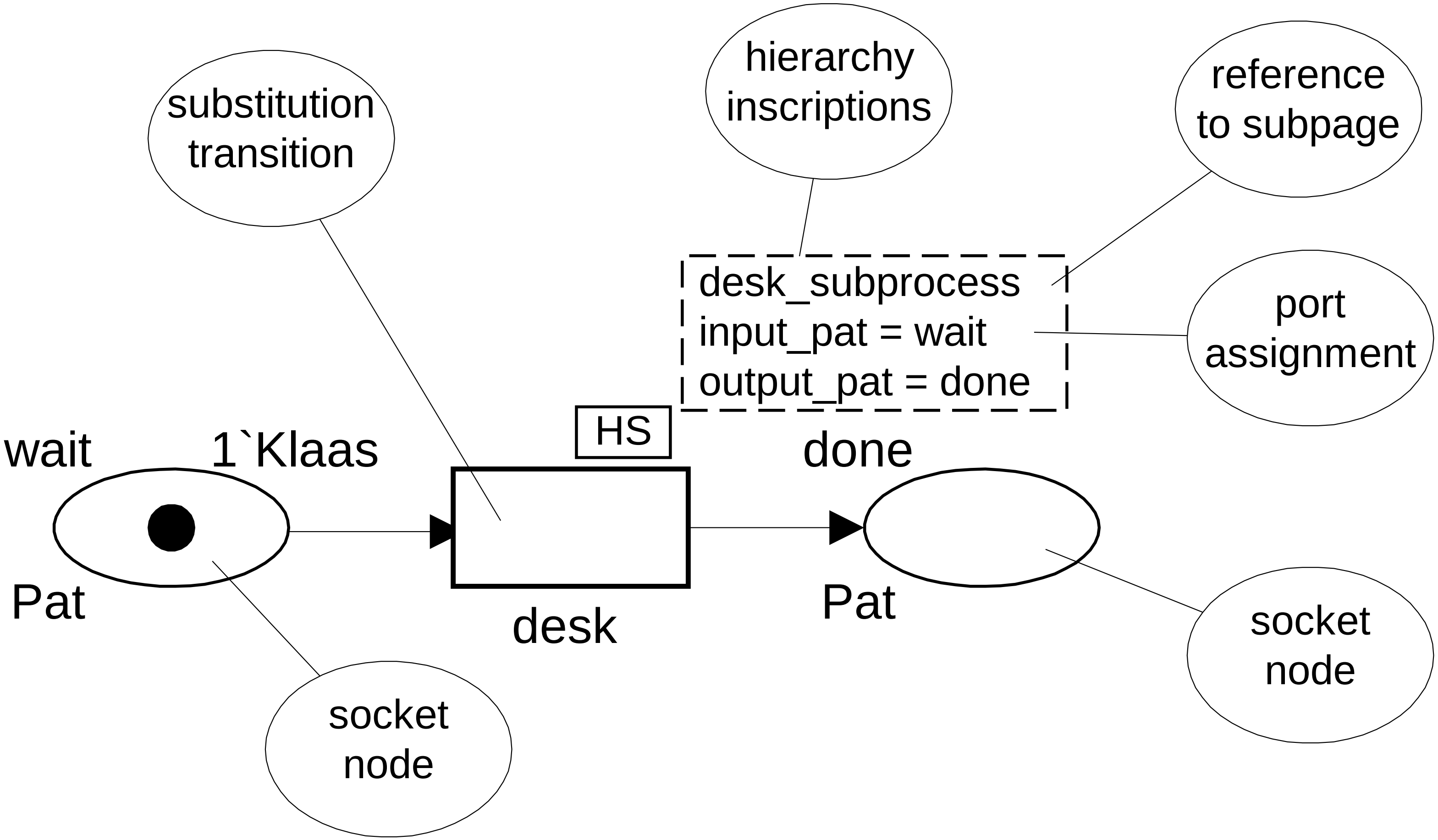


**Let us formalize this in CPN...**

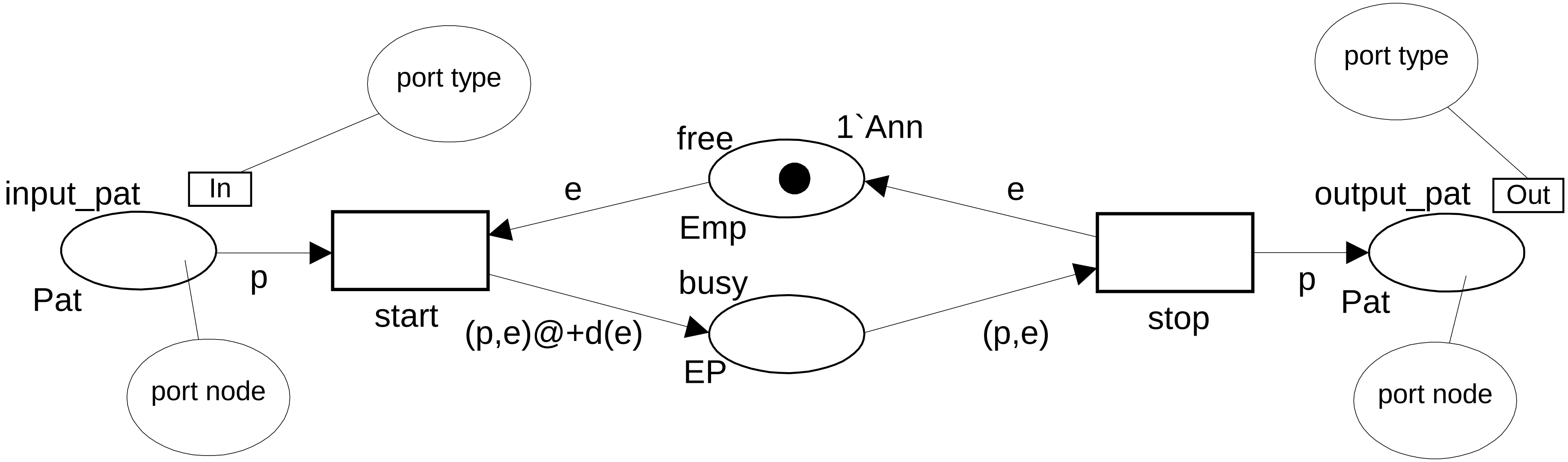
# Hierarchical CPN



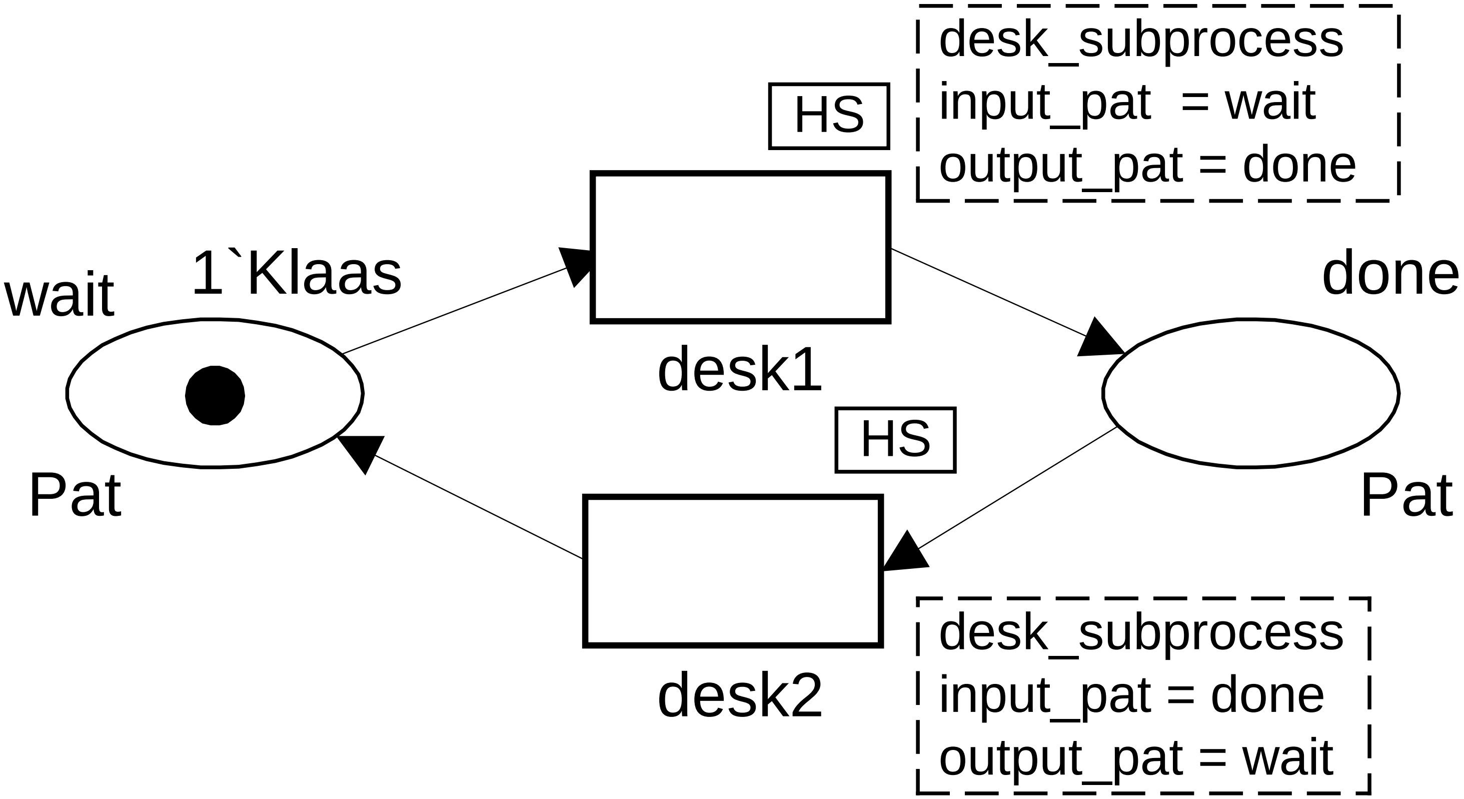
# Superpage: *main*



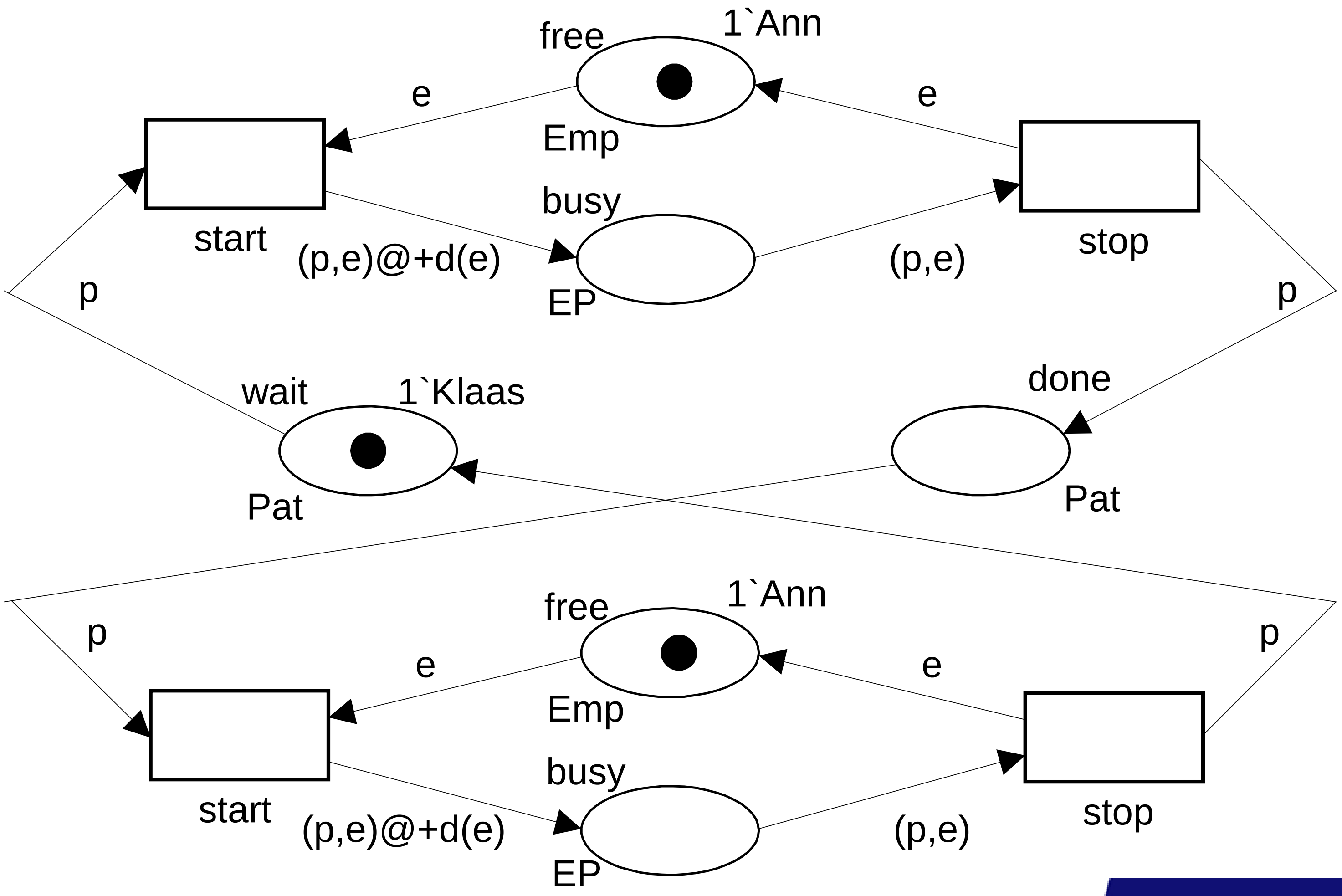
# Subpage: *desk_subprocess*



# One page can have multiple instances (i.e., reuse)

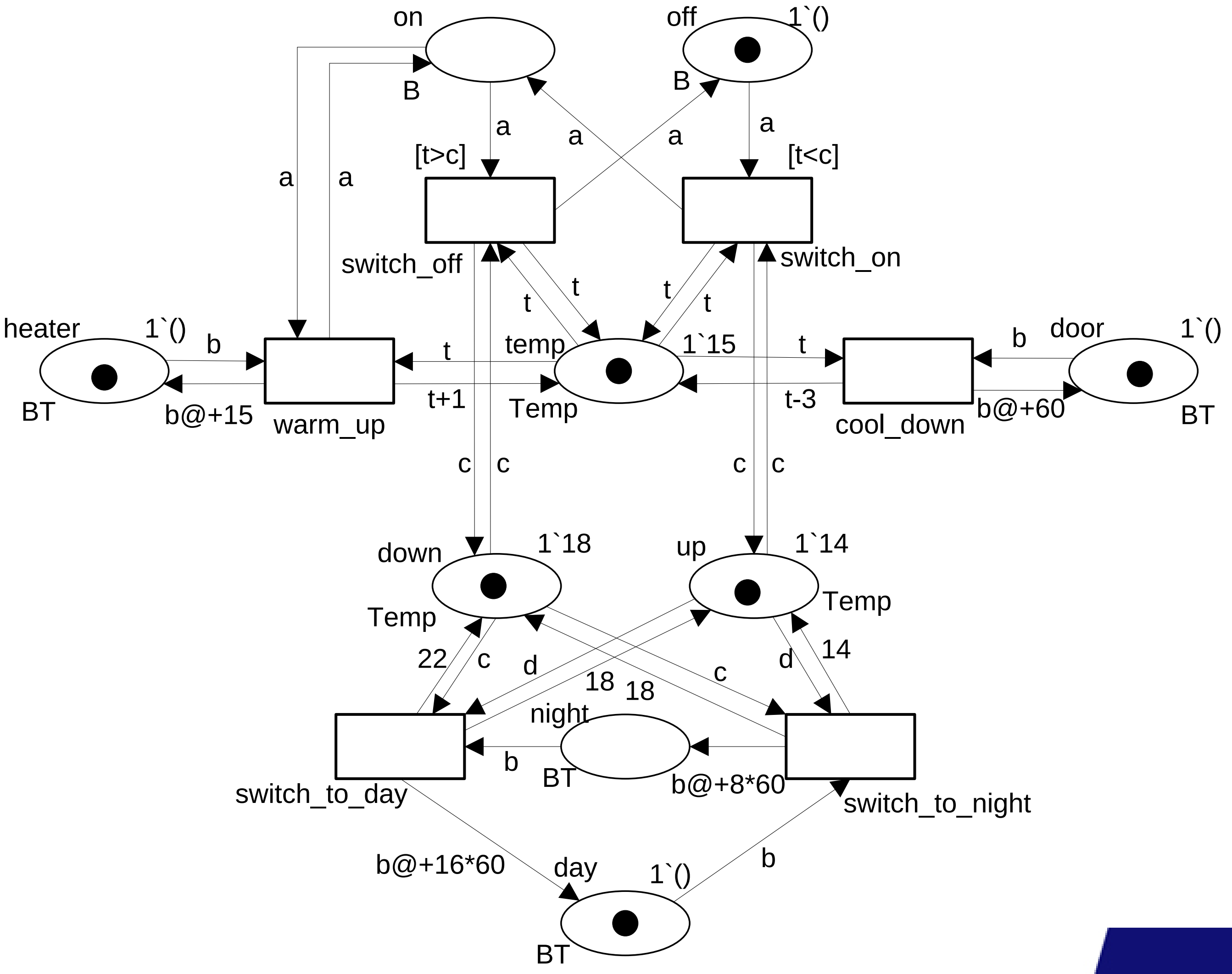


# Semantics: Replace each substitution transition by a copy of the corresponding page



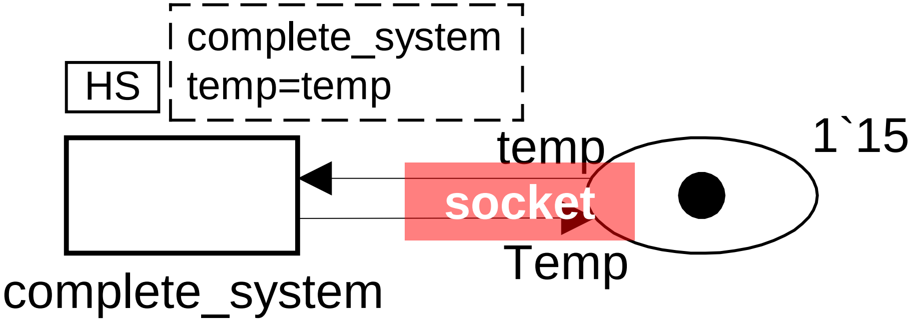
# Thermostat system: Add hierarchy

```
color Temp = string;
color B = unit;
color BT = B timed;
var t:Temp;
var a:B;
var b:BT;
var c,d:Temp;
```

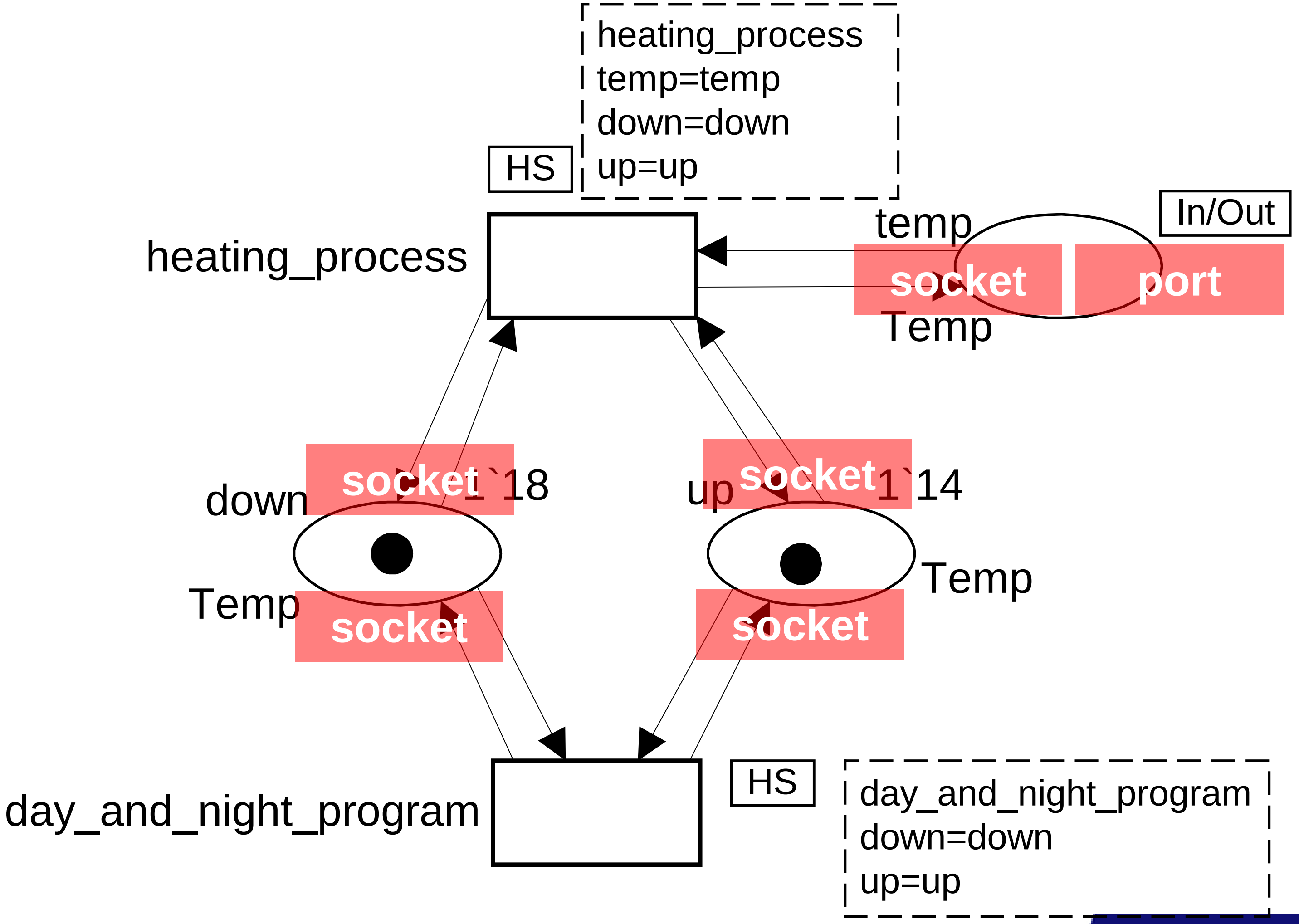


# Top-level page: *main*

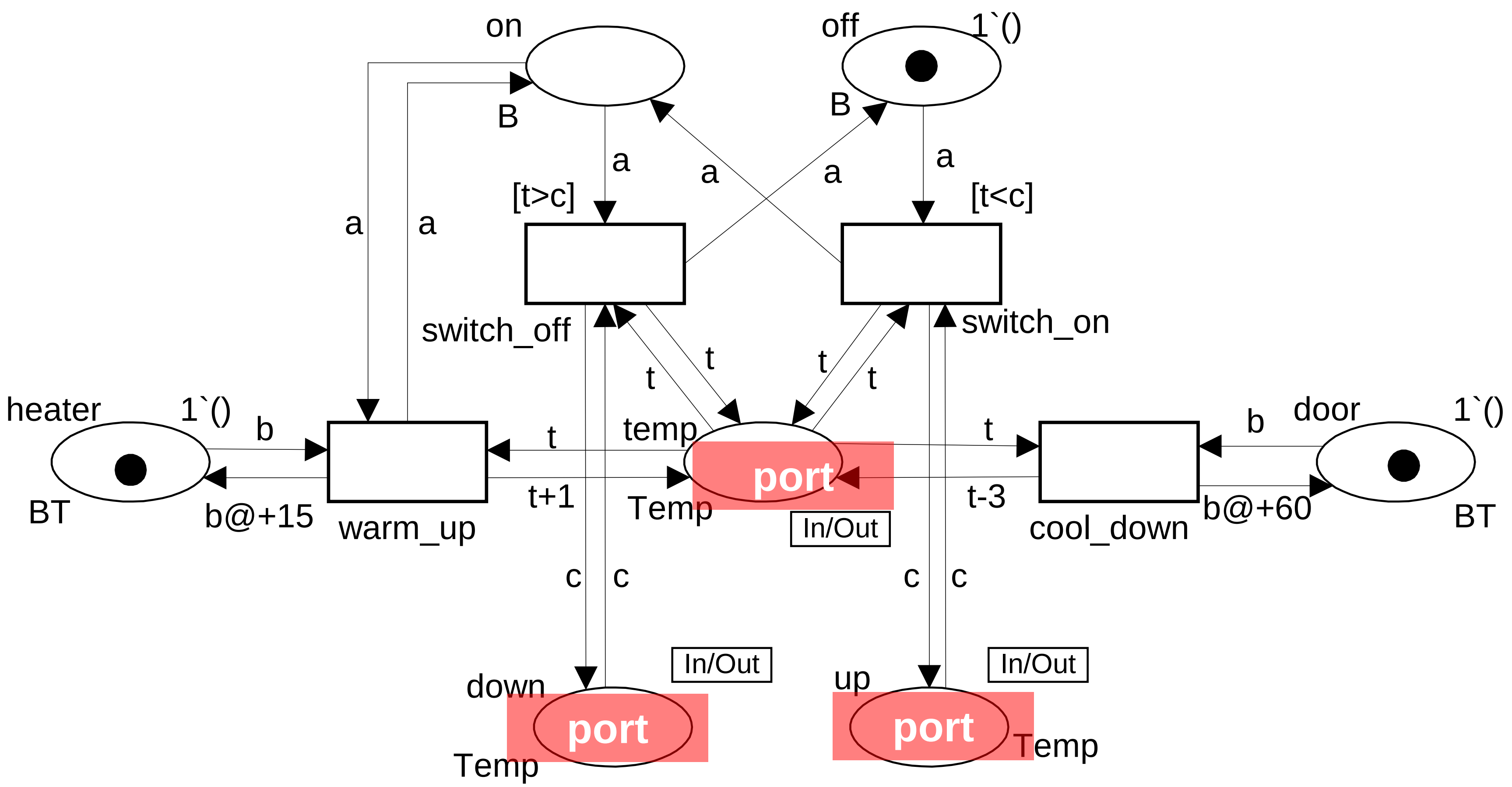
```
color Temp = string;
color B = unit;
color BT = B timed;
var t:Temp;
var a:B;
var b:BT;
var c:Temp;
```



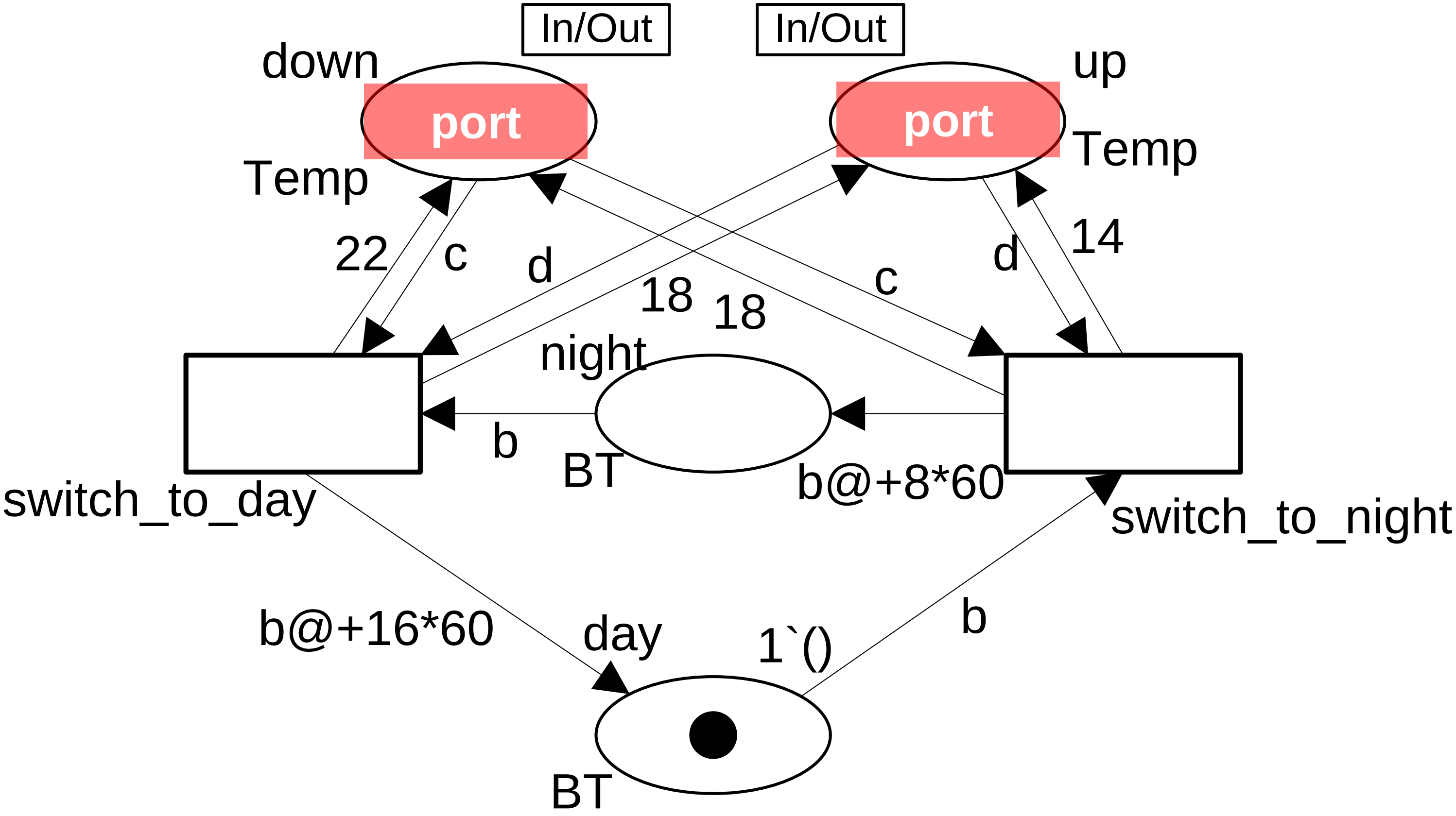
# Subpage: *complete_system*



# Subpage: *heating_process*

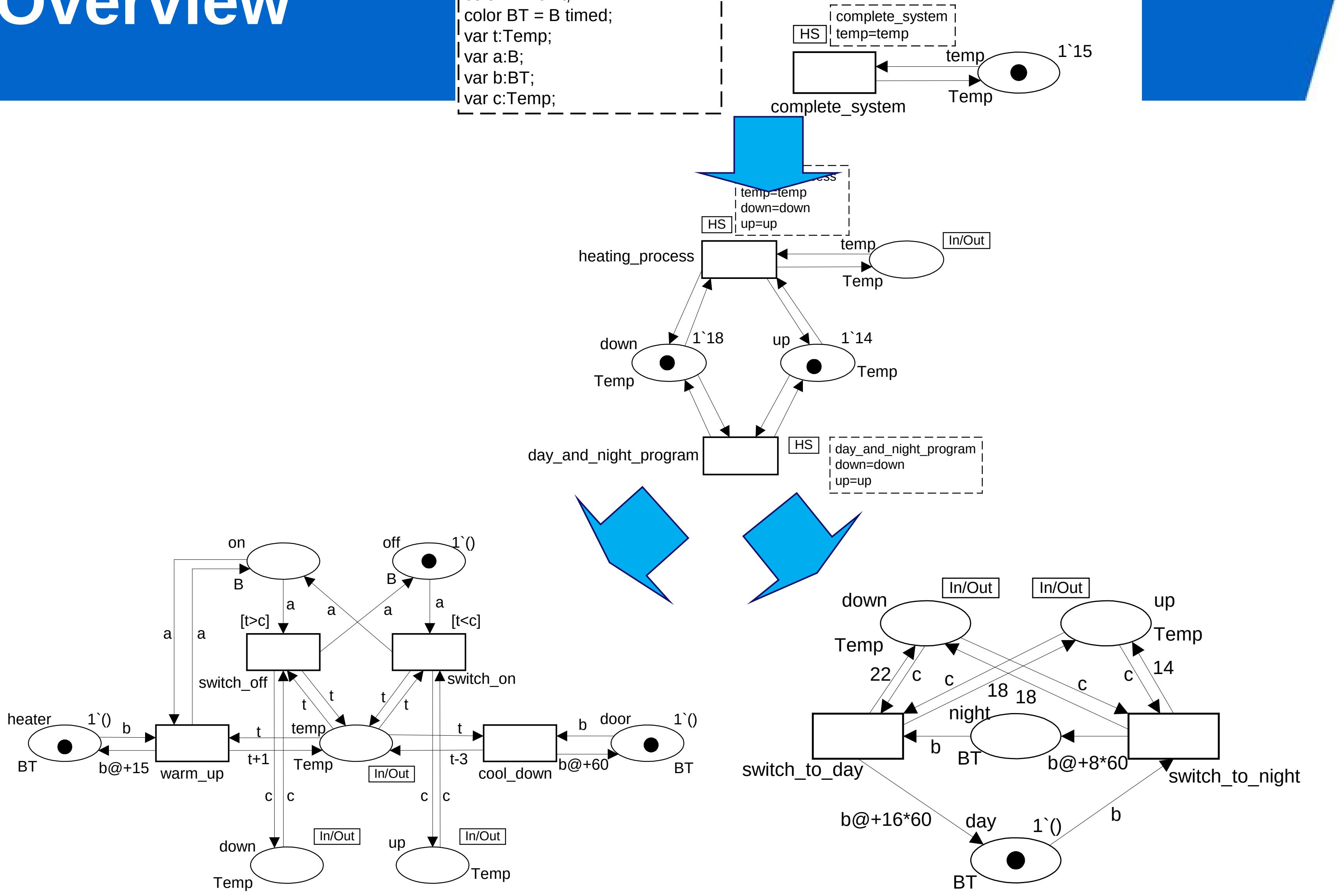


# Subpage: *day_and_night_program*

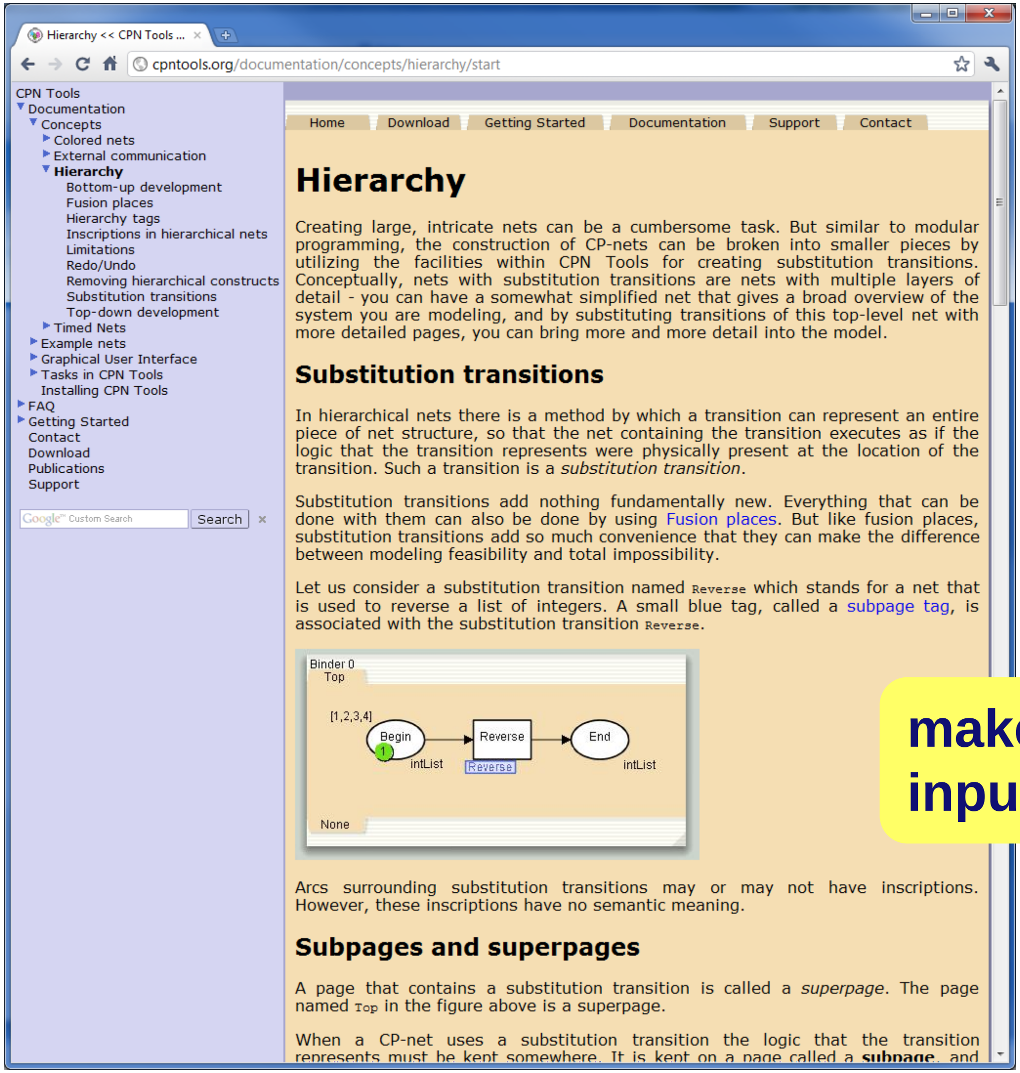


# Overview

```
color Temp = string;
color B = unit;
color BT = B timed;
var t:Temp;
var a:B;
var b:BT;
var c:Temp;
```



# CPN Tools



Hierarchy << CPN Tools ...
cpntools.org/documentation/concepts/hierarchy/start

CPN Tools
- Documentation
  - Concepts
    - Colored nets
    - External communication
    - **Hierarchy**
      - Bottom-up development
      - Fusion places
      - Hierarchy tags
      - Inscriptions in hierarchical nets
      - Limitations
      - Redo/Undo
      - Removing hierarchical constructs
      - Substitution transitions
      - Top-down development
    - Timed Nets
  - Example nets
  - Graphical User Interface
  - Tasks in CPN Tools
  - Installing CPN Tools
- FAQ
- Getting Started
- Contact
- Download
- Publications
- Support

## Hierarchy

Creating large, intricate nets can be a cumbersome task. But similar to modular programming, the construction of CP-nets can be broken into smaller pieces by utilizing the facilities within CPN Tools for creating substitution transitions. Conceptually, nets with substitution transitions are nets with multiple layers of detail - you can have a somewhat simplified net that gives a broad overview of the system you are modeling, and by substituting transitions of this top-level net with more detailed pages, you can bring more and more detail into the model.

### Substitution transitions

In hierarchical nets there is a method by which a transition can represent an entire piece of net structure, so that the net containing the transition executes as if the logic that the transition represents were physically present at the location of the transition. Such a transition is a *substitution transition*.

Substitution transitions add nothing fundamentally new. Everything that can be done with them can also be done by using Fusion places. But like fusion places, substitution transitions add so much convenience that they can make the difference between modeling feasibility and total impossibility.

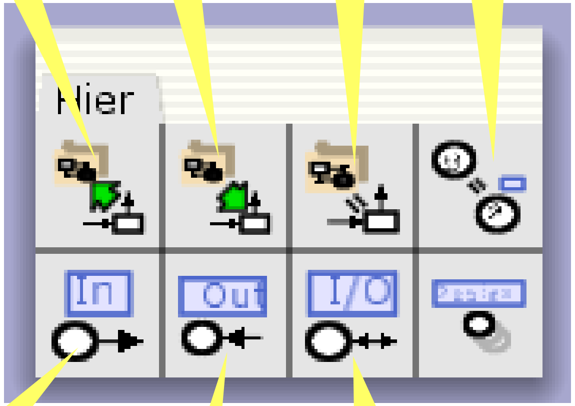
Let us consider a substitution transition named `Reverse` which stands for a net that is used to reverse a list of integers. A small blue tag, called a subpage tag, is associated with the substitution transition `Reverse`.

Arcs surrounding substitution transitions may or may not have inscriptions. However, these inscriptions have no semantic meaning.

### Subpages and superpages

A page that contains a substitution transition is called a *superpage*. The page named `Top` in the figure above is a superpage.

When a CP-net uses a substitution transition the logic that the transition represents must be kept somewhere. It is kept on a page called a **subpage**, and



move to subpage

assign subpage

unfold

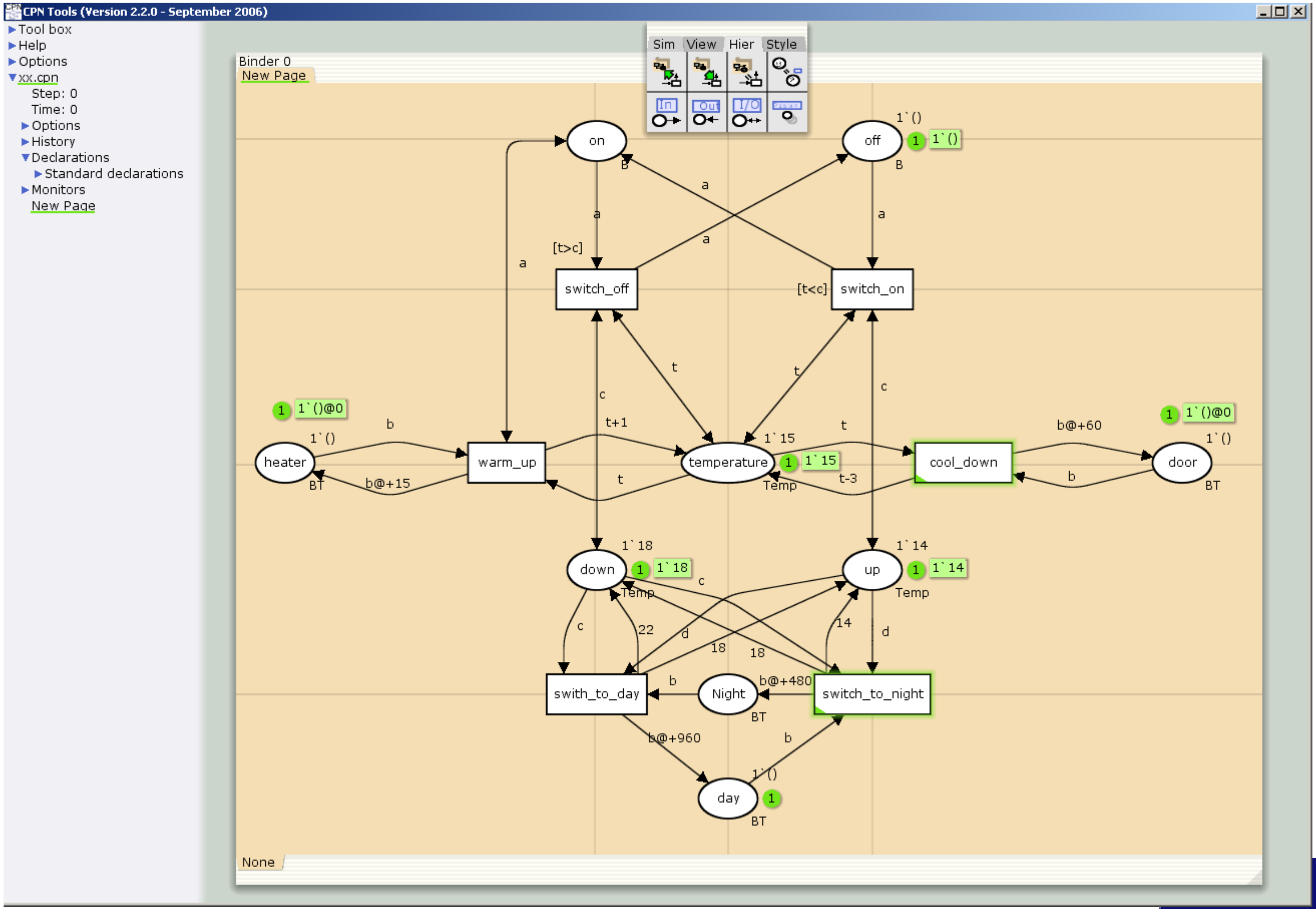
connect

make input port

make output port

make I/O port

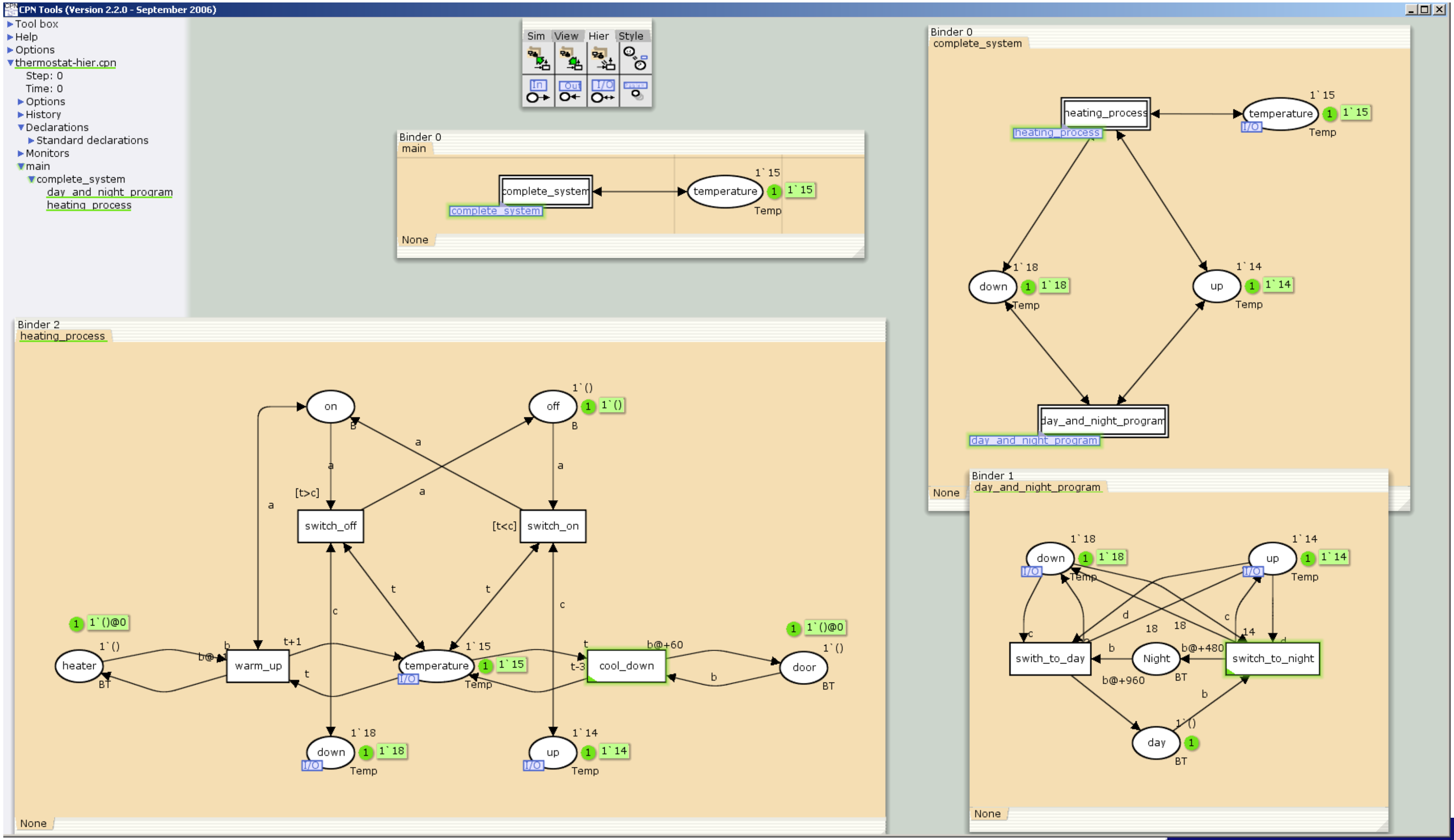
# CPN Tools (flat model)



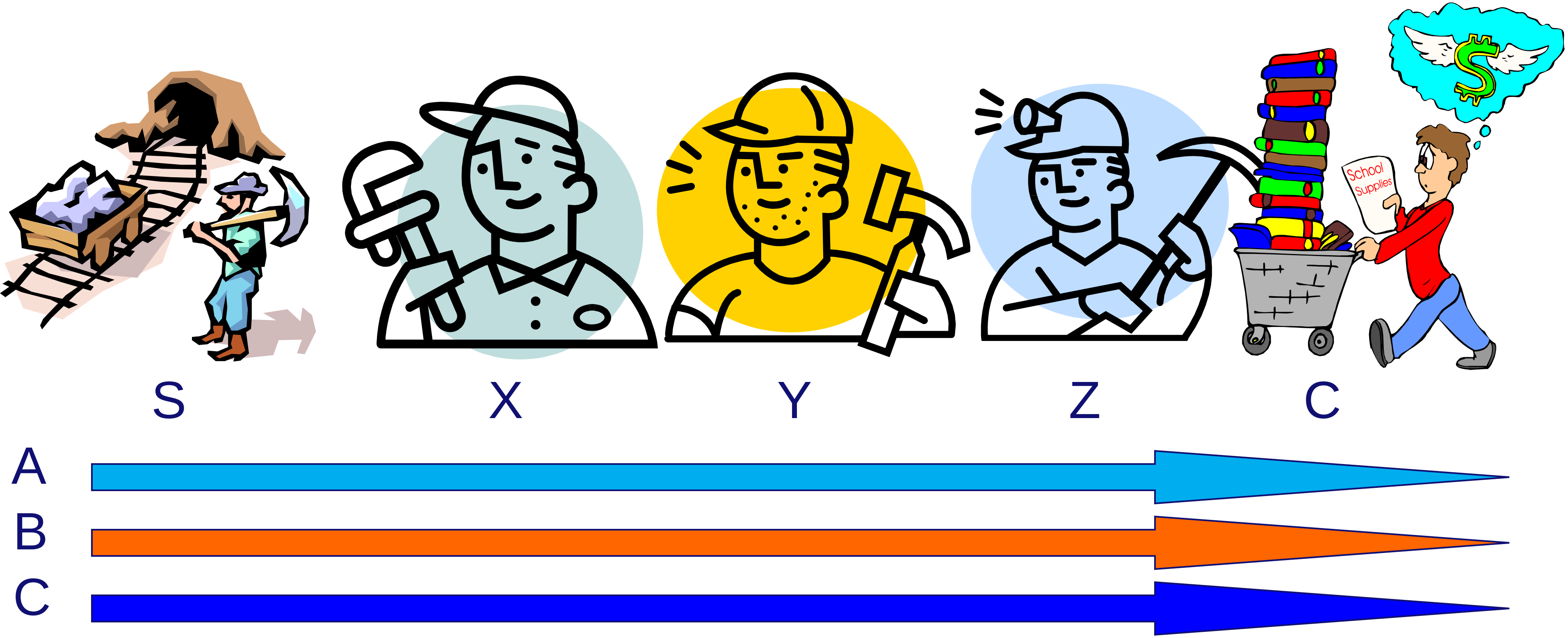
CPN Tools (Version 2.2.0 - September 2006)

- ▶Tool box
- ▶Help
- ▶Options
- ▼xx.cpn
  - Step: 0
  - Time: 0
  - ▶Options
  - ▶History
  - ▼Declarations
    - ▶Standard declarations
  - ▶Monitors
  - New Page

# With hierarchy



# Example: Production system



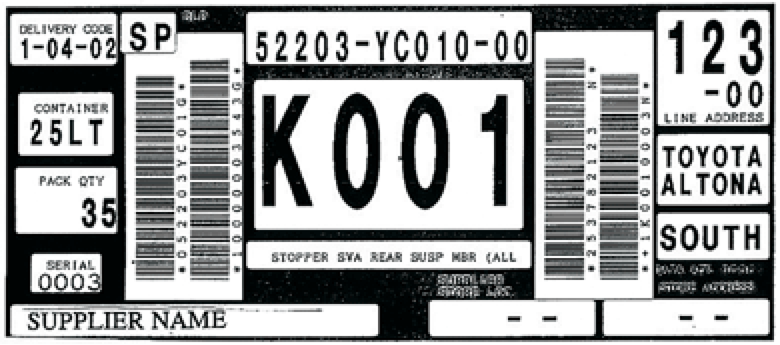
S X Y Z C

A

B

C

# Kanban


DELIVERY CODE 1-04-02

SP

52203-YC010-00

123 -00 LINE ADDRESS

CONTAINER 25LT

K001

TOYOTA ALTONA

PACK QTY 35

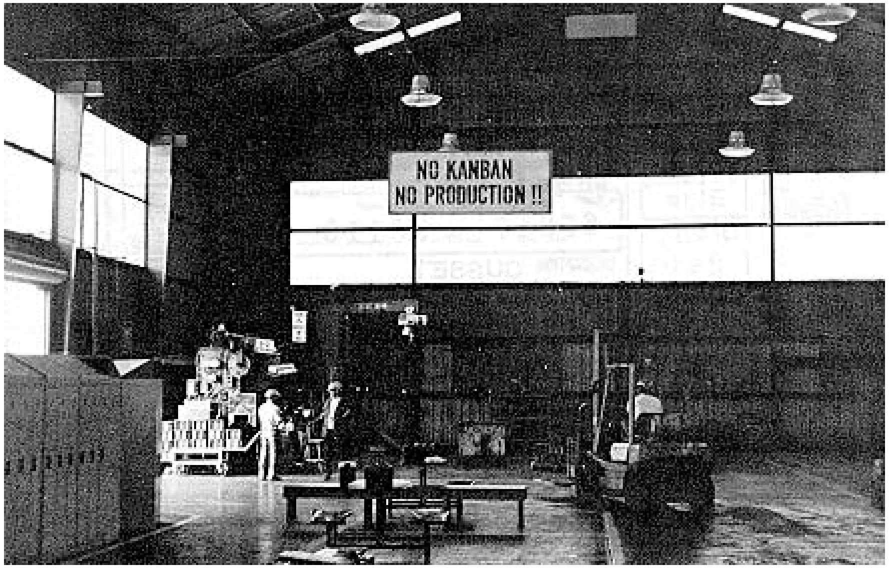
SOUTH

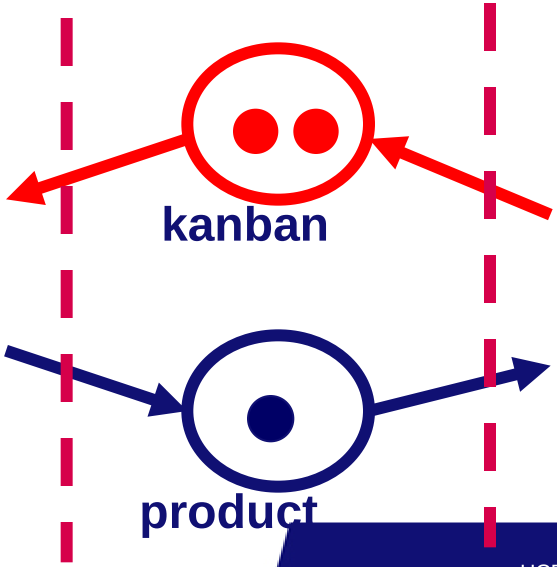
SERIAL 0003

STOPPER SVA REAR SUSP MBR (ALL

SUPPLIER NAME

- Kanbans are a means to achieve JIT (Just-In-Time).
- Japanise word for card.
- Attributed to Taiichi Ohno (the father of the Toyota Production System)





# Two beer kanban

# Data

|  | X | Y | Z |
|---|---|---|---|
| A | **2** | **5** | **8** |
| B | **3** | **6** | **9** |
| C | **4** | **7** | **1** |

*Processing times*

| | |
|---|---|
| X | **2** |
| Y | **3** |
| Z | **4** |

*Resources per work center*

| | |
|---|---|
| A | **2** |
| B | **1** |
| C | **2** |

*Replenishment lead times*

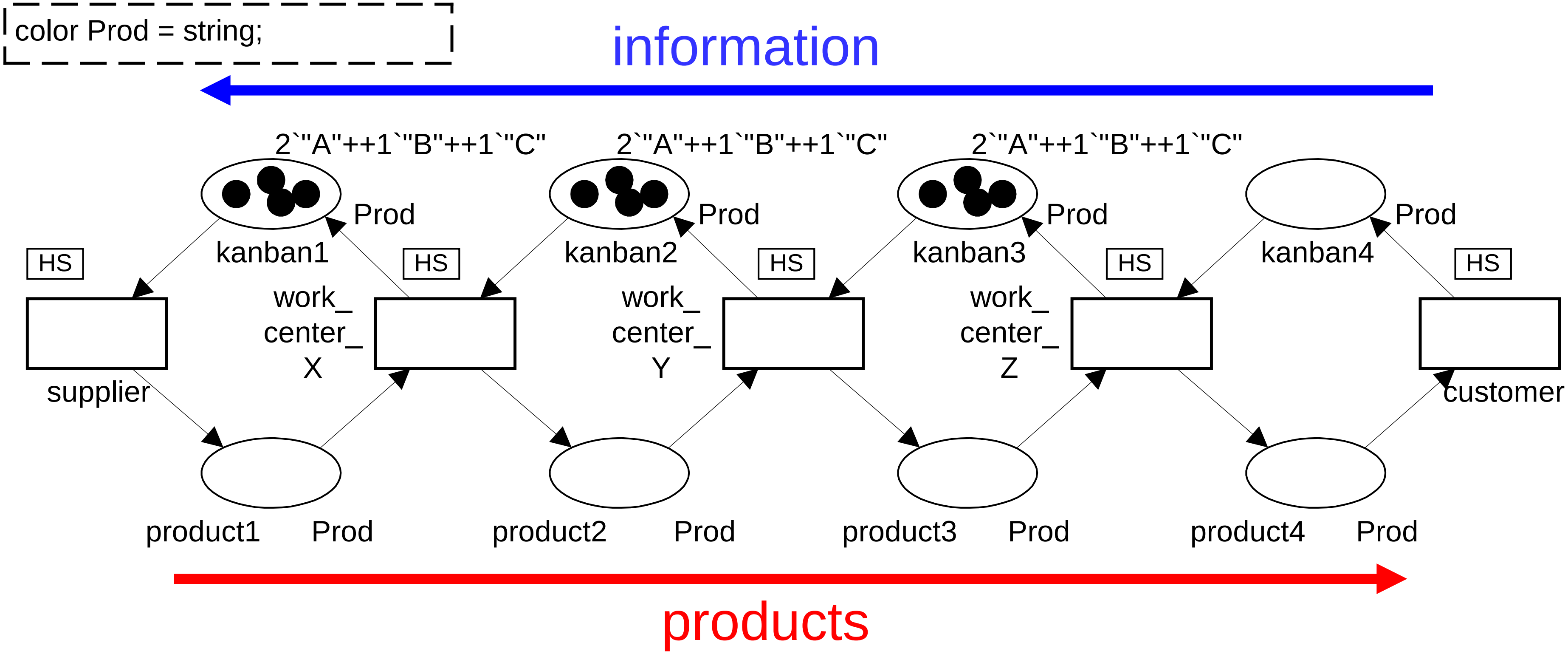
| | |
|---|---|
| A | **2** |
| B | **1** |
| C | **1** |

*Kanbans in-between work centers*

| | |
|---|---|
| A | **7** |
| B | **9** |
| C | **8** |

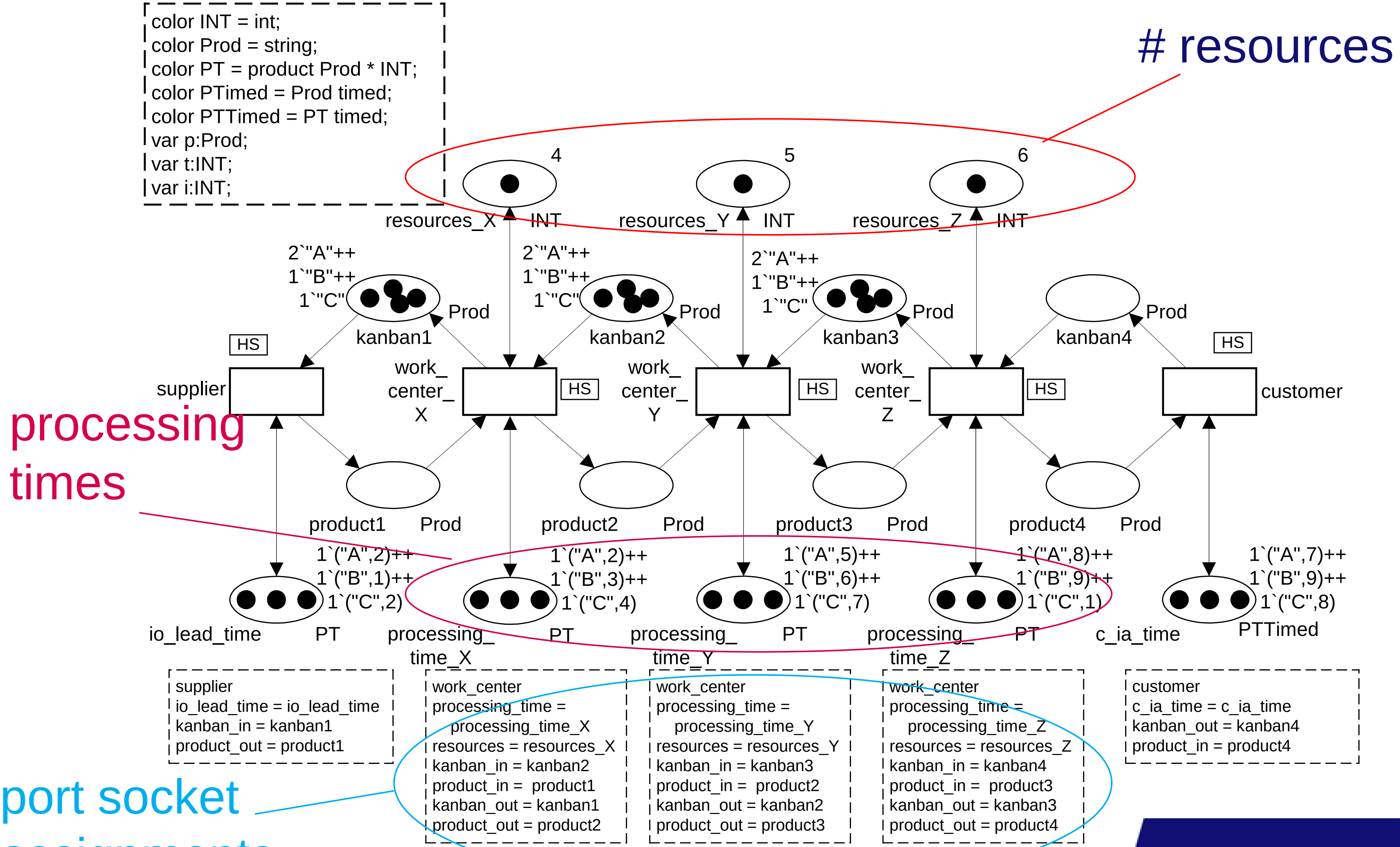
*Time in-between subsequent orders*

# Initial model



**How to configure each work center if they share the same page definition???**

# Top level page: *main*



```
color INT = int;
color Prod = string;
color PT = product Prod * INT;
color PTimed = Prod timed;
color PTTimed = PT timed;
var p:Prod;
var t:INT;
var i:INT;
```

# resources

processing times

supplier
io_lead_time = io_lead_time
kanban_in = kanban1
product_out = product1

work_center
processing_time =
processing_time_X
resources = resources_X
kanban_in = kanban2
product_in = product1
kanban_out = kanban1
product_out = product2

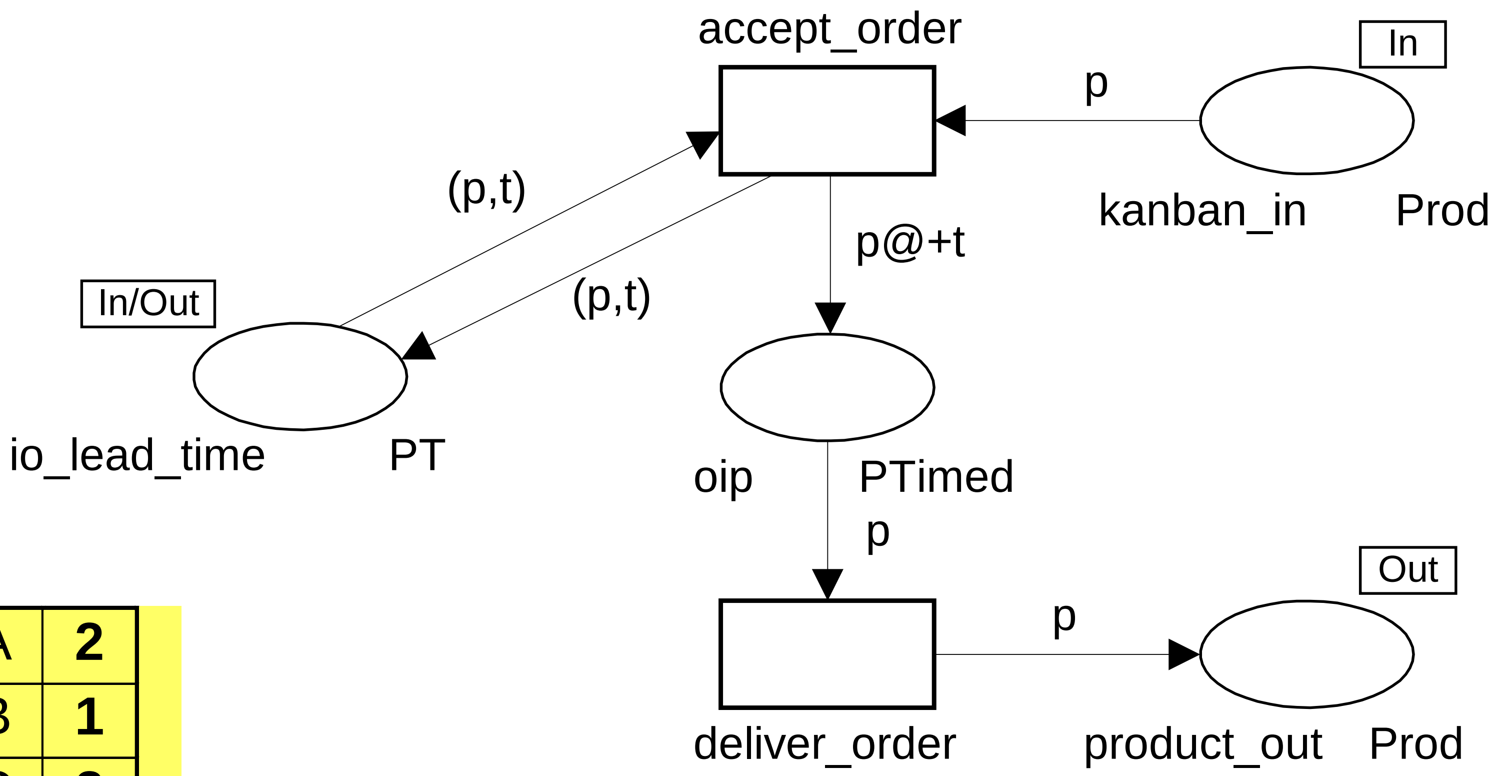
work_center
processing_time =
processing_time_Y
resources = resources_Y
kanban_in = kanban3
product_in = product2
kanban_out = kanban2
product_out = product3

work_center
processing_time =
processing_time_Z
resources = resources_Z
kanban_in = kanban4
product_in = product3
kanban_out = kanban3
product_out = product4

customer
c_ia_time = c_ia_time
kanban_out = kanban4
product_in = product4

port socket assignments

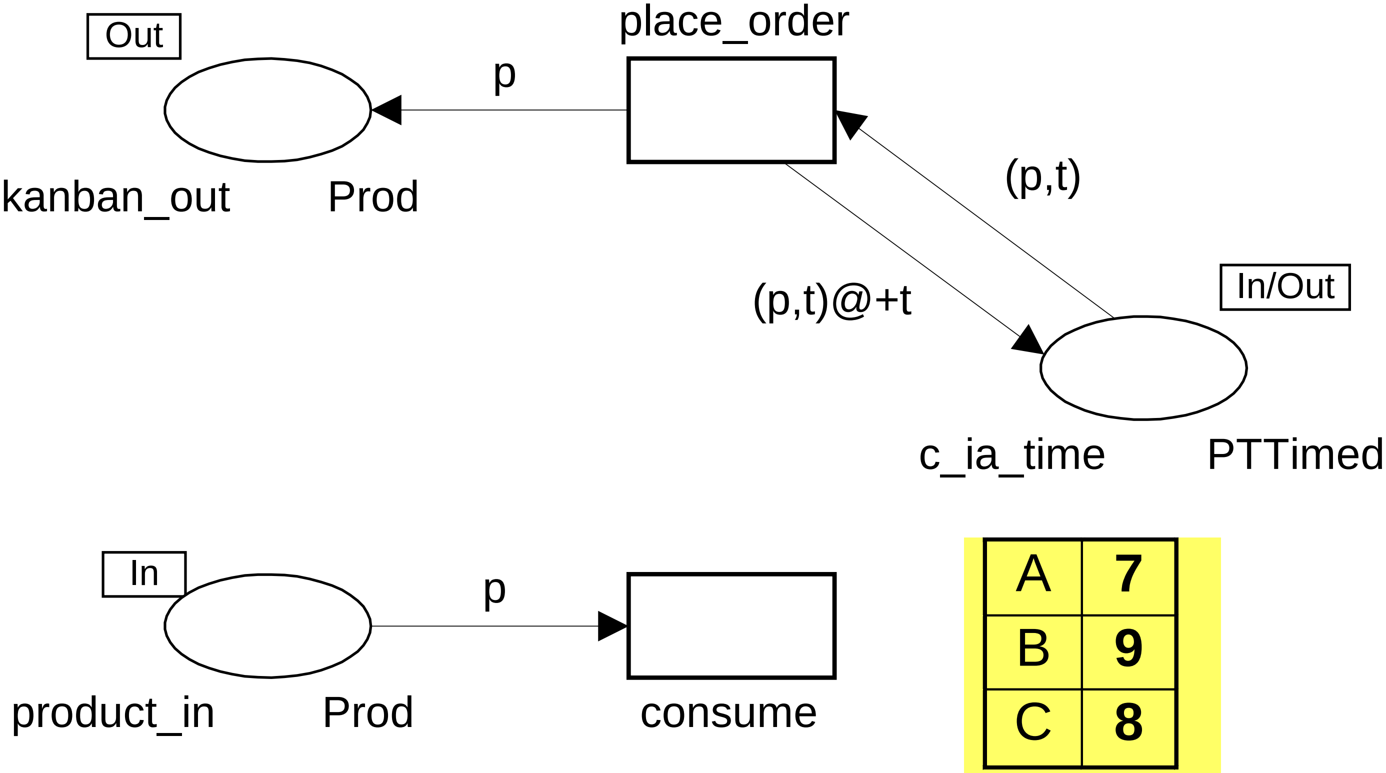
# Sub page: *supplier*



| | |
|---|---|
| A | **2** |
| B | **1** |
| C | **2** |

*Replenishment lead times*

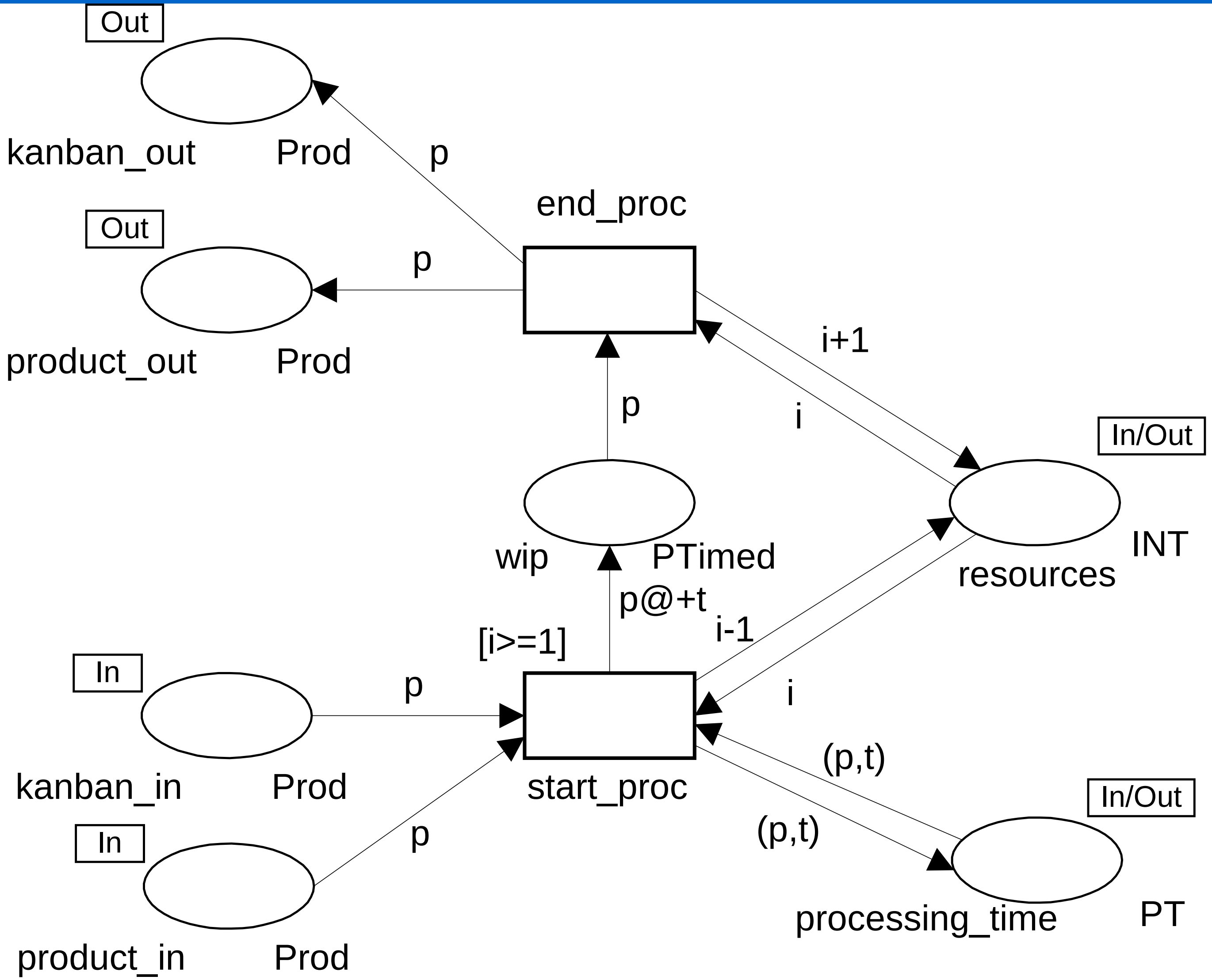
# Sub page: *customer*



| | |
|---|---|
| A | **7** |
| B | **9** |
| C | **8** |

*Time in-between subsequent orders*

# Sub page: *work_center*



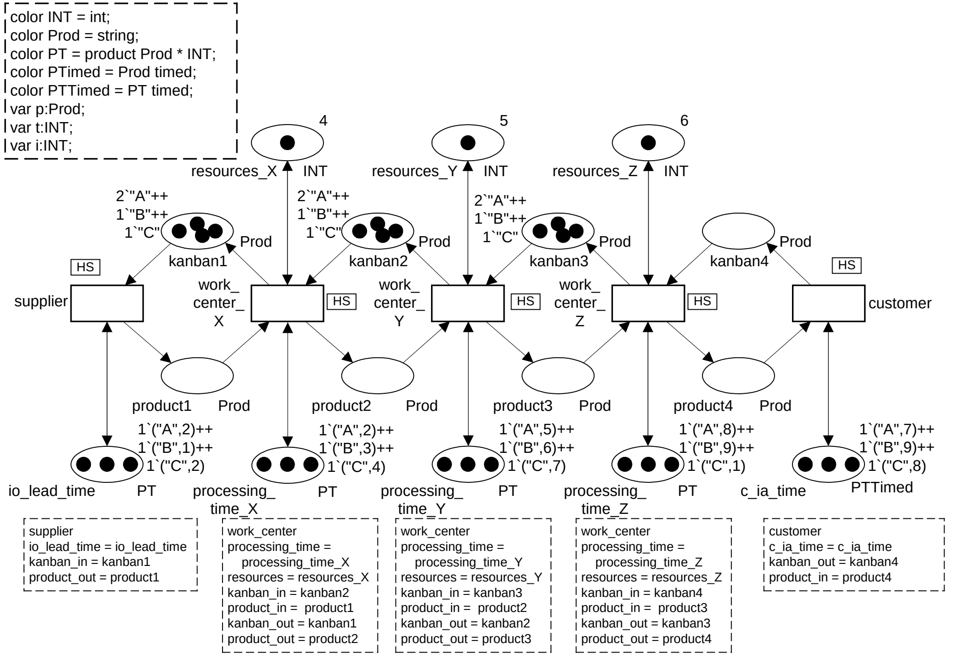
| | |
|---|---|
| X | **2** |
| Y | **3** |
| Z | **4** |

*Resources per work center*

| | X | Y | Z |
|---|---|---|---|
| A | **2** | **5** | **8** |
| B | **3** | **6** | **9** |
| C | **4** | **7** | **1** |

*Processing times*

# Overview



```
color INT = int;
color Prod = string;
color PT = product Prod * INT;
color PTimed = Prod timed;
color PTTimed = PT timed;
var p:Prod;
var t:INT;
var i:INT;
```

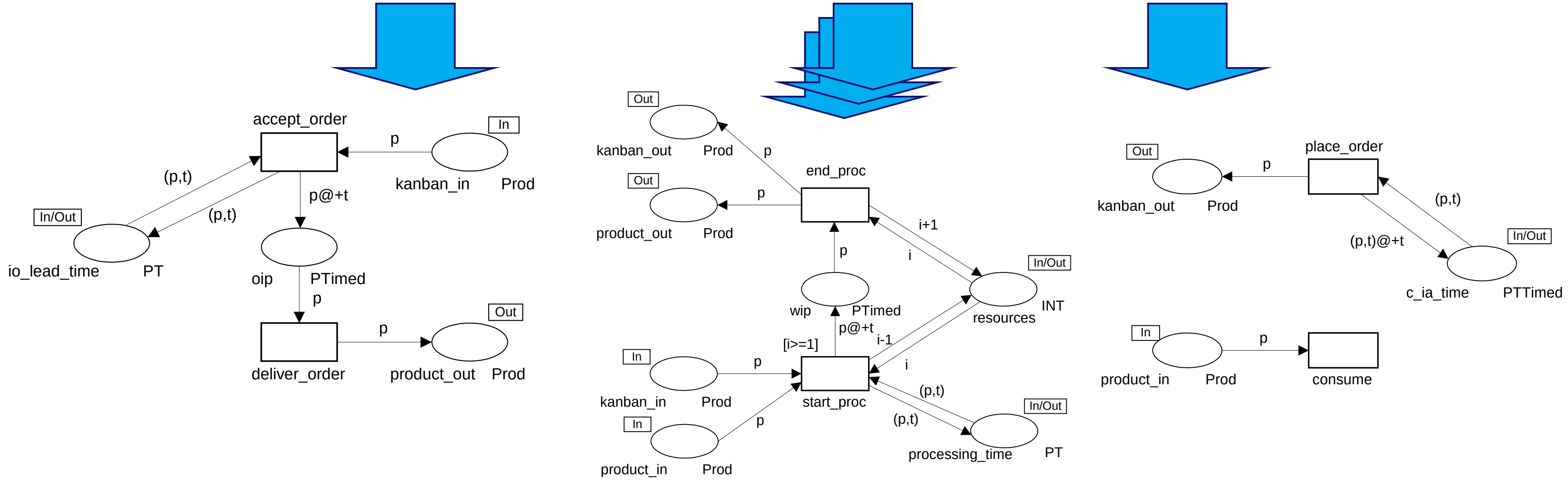
supplier
io_lead_time = io_lead_time
kanban_in = kanban1
product_out = product1

work_center
processing_time =
processing_time_X
resources = resources_X
kanban_in = kanban2
product_in = product1
kanban_out = kanban1
product_out = product2

work_center
processing_time =
processing_time_Y
resources = resources_Y
kanban_in = kanban3
product_in = product2
kanban_out = kanban2
product_out = product3

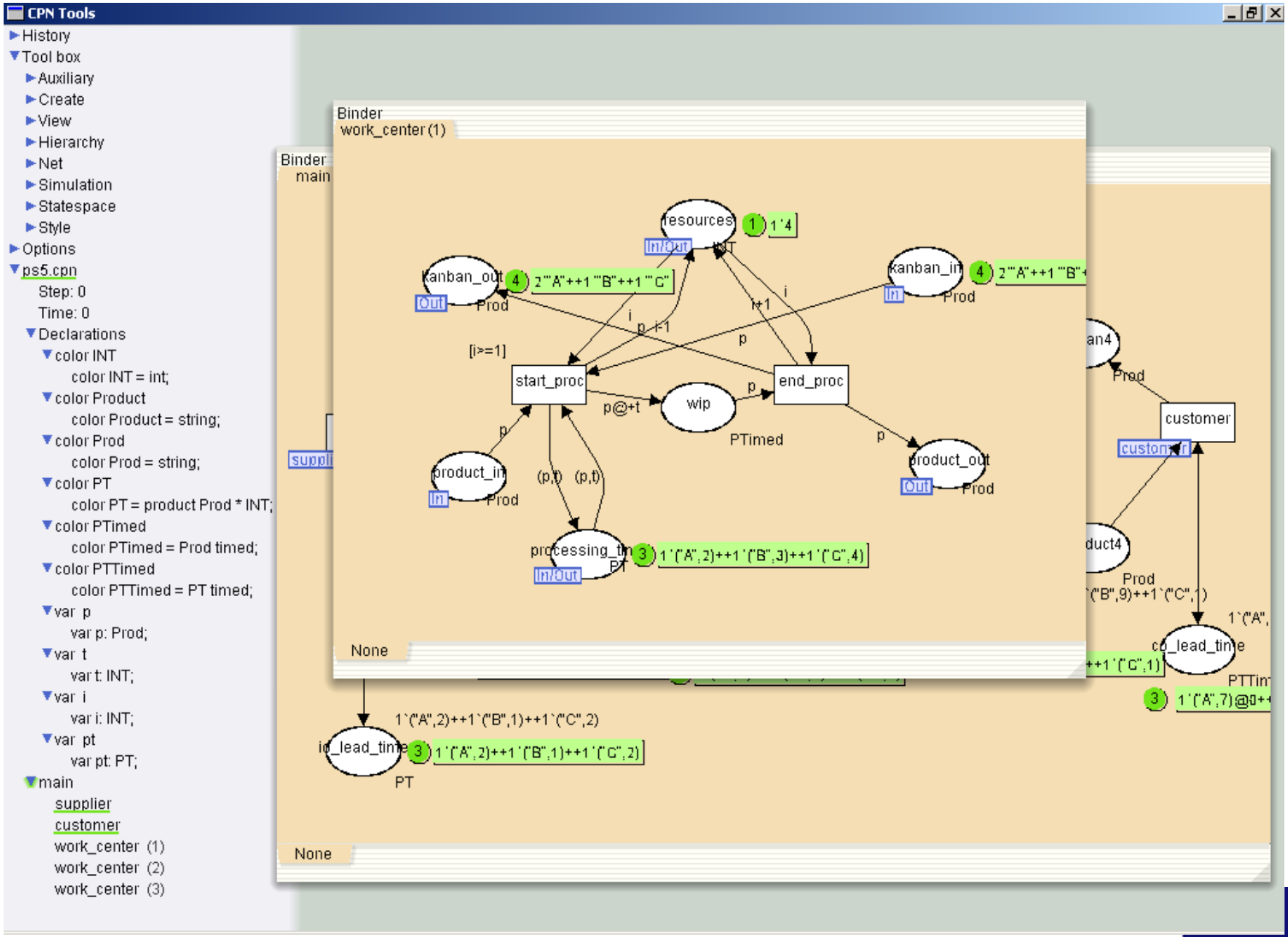
work_center
processing_time =
processing_time_Z
resources = resources_Z
kanban_in = kanban4
product_in = product3
kanban_out = kanban3
product_out = product4

customer
c_ia_time = c_ia_time
kanban_out = kanban4
product_in = product4



*One page definition,*
*three page instances!!*

# CPN tools



```
color INT
    color INT = int;
color Product
    color Product = string;
color Prod
    color Prod = string;
color PT
    color PT = product Prod * INT;
color PTimed
    color PTimed = Prod timed;
color PTTimed
    color PTTimed = PT timed;
var p
    var p: Prod;
var t
    var t: INT;
var i
    var i: INT;
var pt
    var pt: PT;
```

# Inventory strategies: from (s,Q)-b to (R,s,S)+b

MY COMPANY IS MOVING TO A "JUST IN TIME" INVENTORY STRATEGY. YOU'LL DELIVER WHEN WE NEED IT.
SO...YOUR SUCCESS DEPENDS ON MY COMPANY DOING WHAT IT PROMISES? YOU HAVE MY DEEPEST SYMPATHY.
I FEEL A SHARP, STABBING PAIN IN MY CHEST.
AND SO IT BEGINS.
www.dilbert.com scottadams@aol.com
3/14/03 © 2003 United Feature Syndicate, Inc.

# Application: Modeling logistic processes

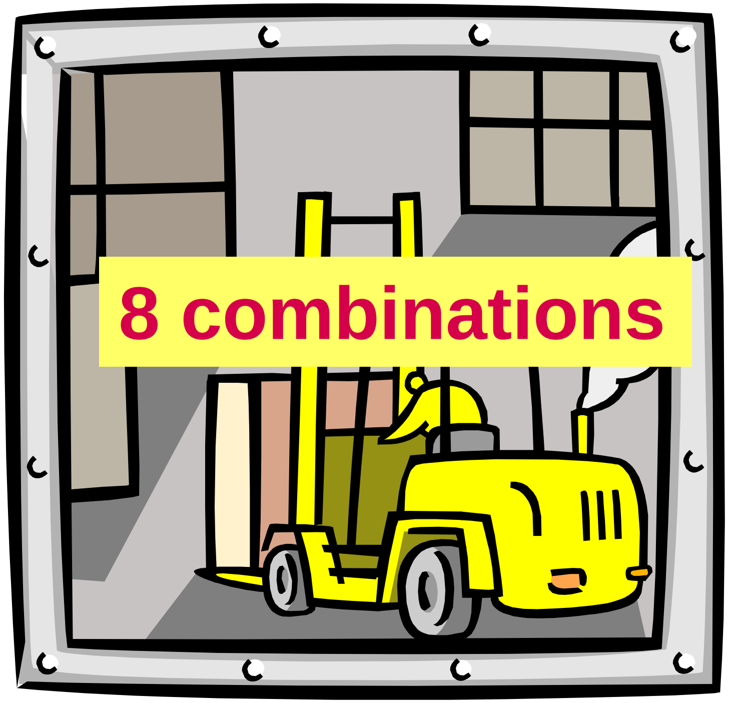
**periodic or instant: R,s or s**

(s,S)-b

(s,Q)-b

(R,s,S)-b

(s,S)+b



**8 combinations**

(s,Q)+b

(R,s,S)+b

(R,S)-b

(R,S)+b

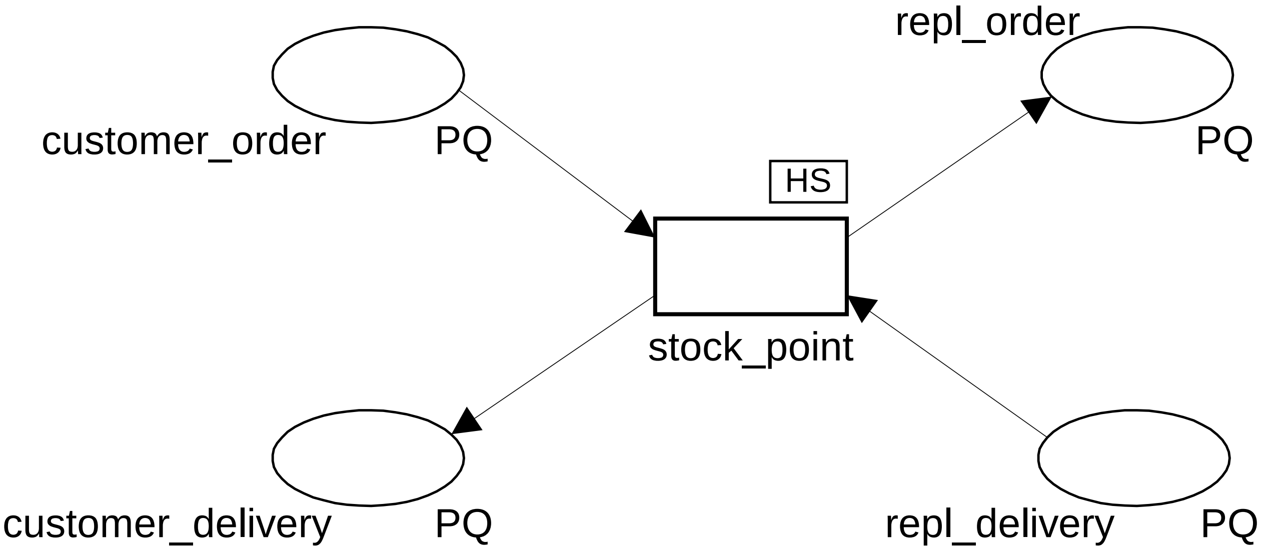
**fixed or order-up-to-level: Q or S**

**backorders or not: -b or +b**

# External interface of a stock point

**orders**

```
color Product = string;
color Quantity = int;
color PQ = product Product * Quantity;
```



**goods**

(For simplicity we do not add external configuration places.)

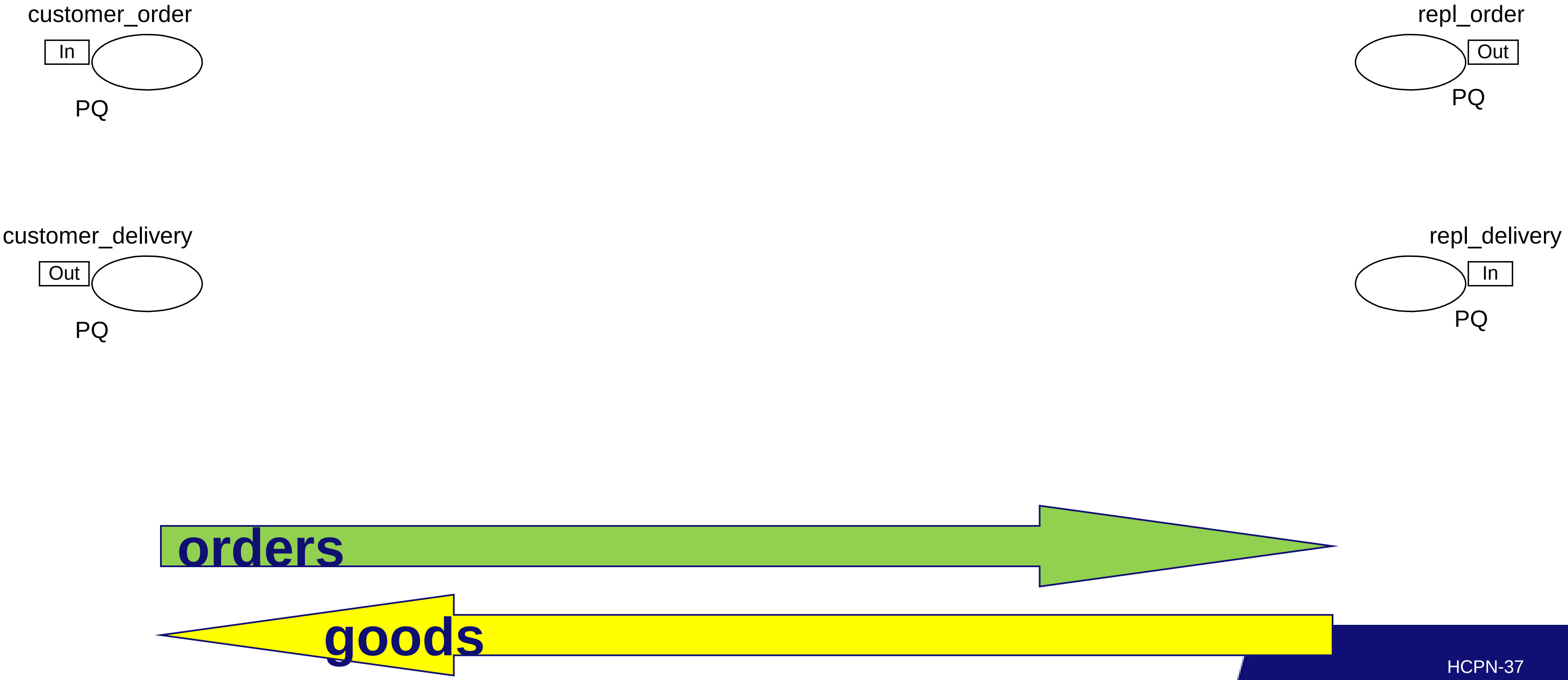
# Inventory policy (s,Q)-b

- Complete lost sales, i.e., no backorders.
- Continuous review, i.e., *R=0*.
- Order point *s.*
- Fixed order quantity *Q.*
- *s* and *Q* determined for each product individually.

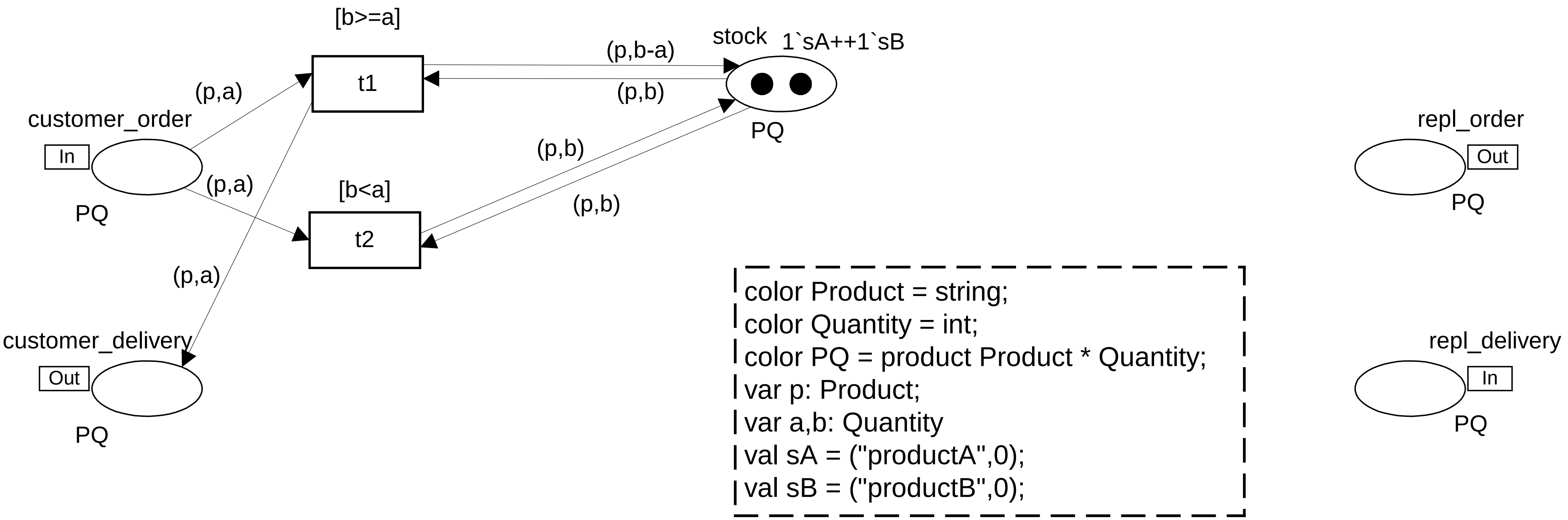
Order point is compared with inventory position, i.e., on-hand stock + ordered – back orders.
(Note that back orders = 0)

# Step 1: Port nodes

```
color Product = string;
color Quantity = int;
color PQ = product Product * Quantity;
```



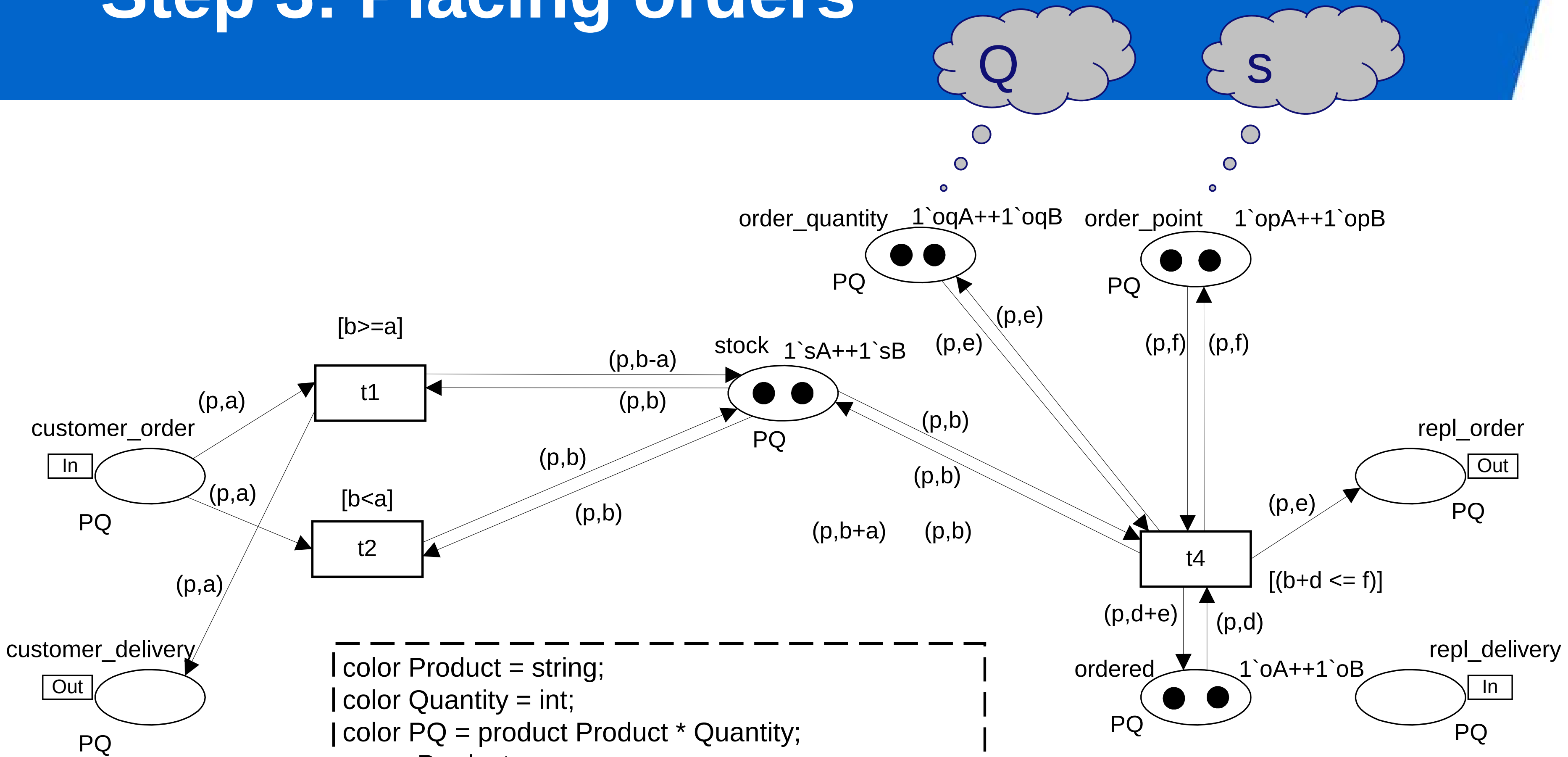
# Step 2: Dealing with customer orders



```
color Product = string;
color Quantity = int;
color PQ = product Product * Quantity;
var p: Product;
var a,b: Quantity
val sA = ("productA",0);
val sB = ("productB",0);
```

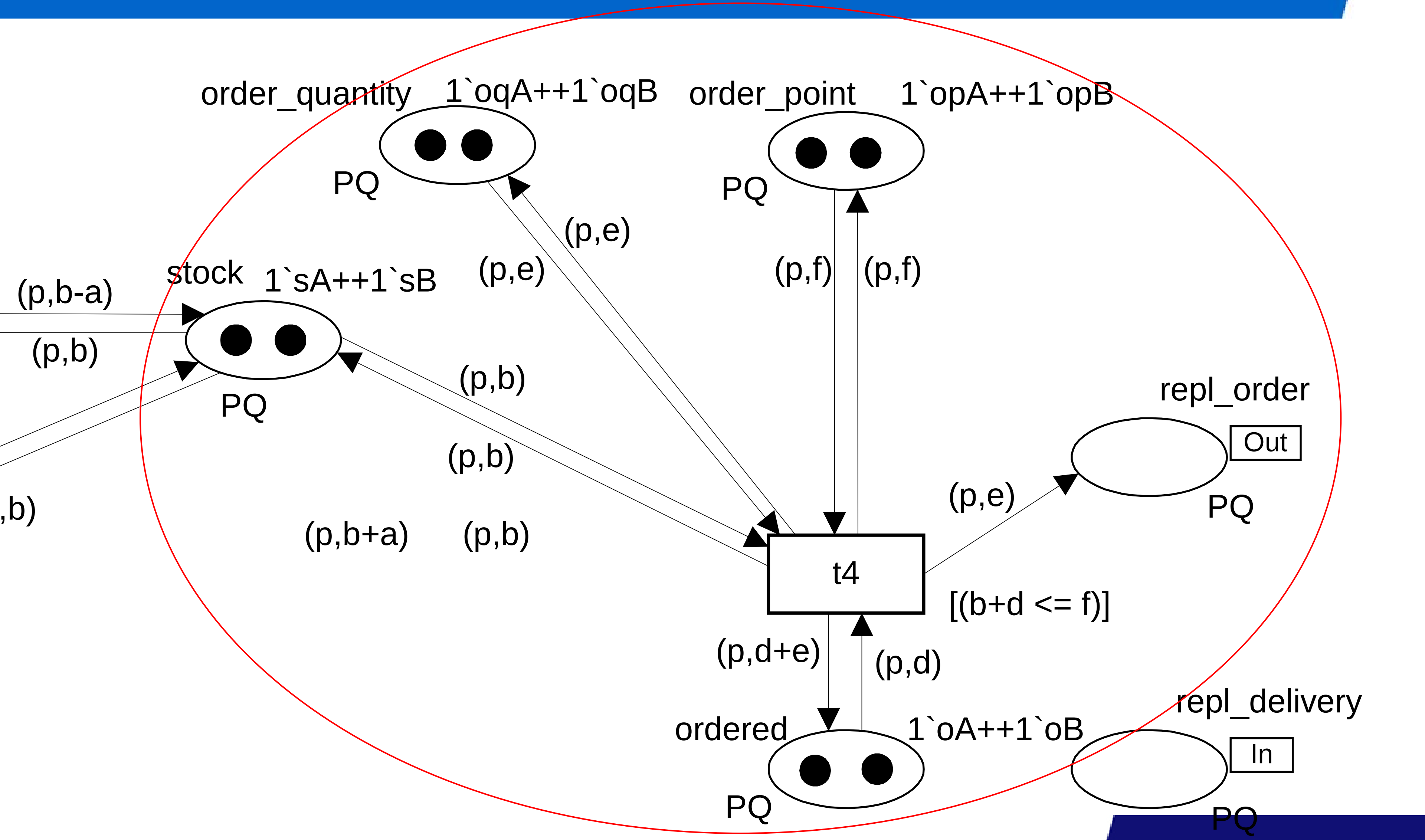
In example, we assume two products: "productA" and "productB". Of course the structure of the model stays the same for any set of products.

# Step 3: Placing orders

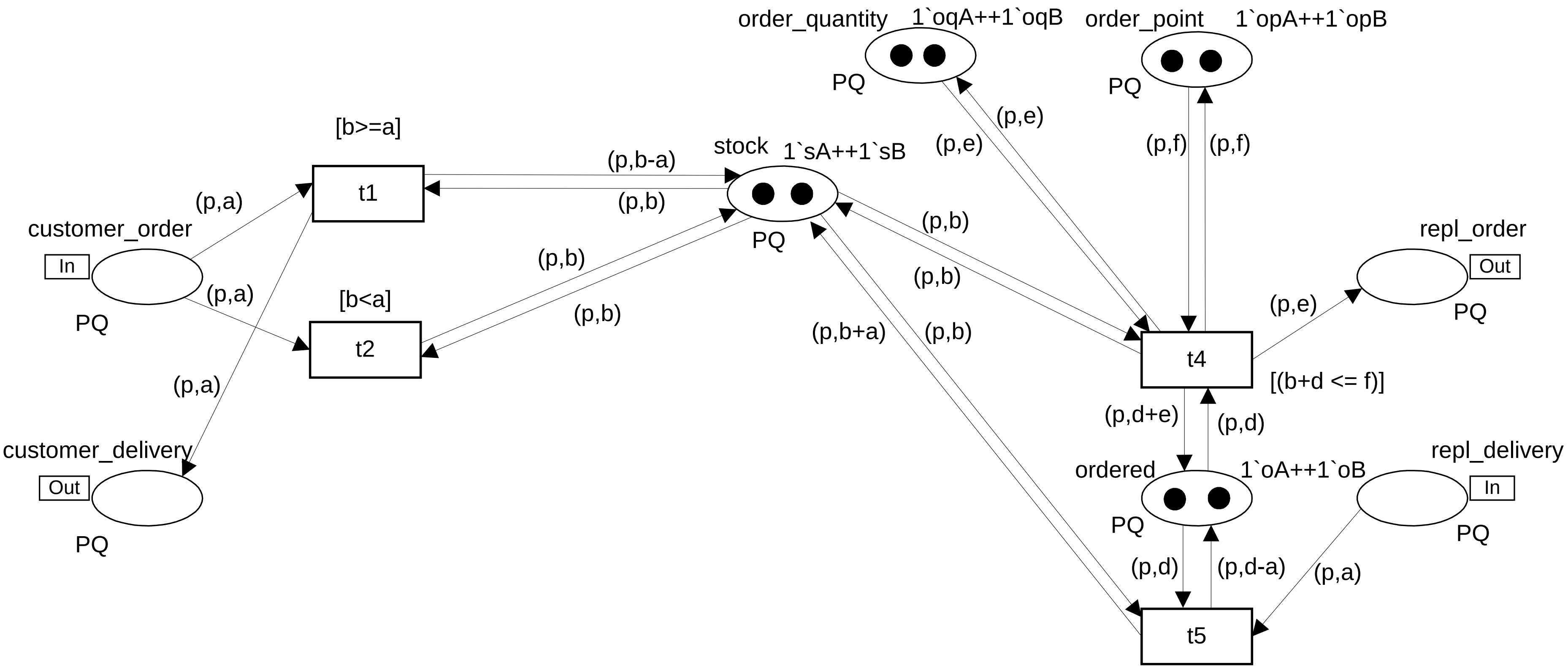


```
color Product = string;
color Quantity = int;
color PQ = product Product * Quantity;
var p: Product;
var a,b,c,d,e,f: Quantity
val sA = ("productA",0);
val sB = ("productB",0);
val oA = ("productA",0);
val oB = ("productB",0);
val oqA = ("productA",150);
val oqB = ("productB",100);
val opA = ("productA",50);
val opB = ("productA",60);
```

# Zoom in


order_quantity
1`oqA++1`oqB
PQ
order_point
1`opA++1`opB
PQ
(p,e)
(p,e)
(p,f)
(p,f)
stock
1`sA++1`sB
(p,b-a)
(p,b)
PQ
(p,b)
(p,b)
,b)
(p,b+a)
(p,b)
repl_order
Out
PQ
(p,e)
t4
[(b+d <= f)]
(p,d+e)
(p,d)
ordered
1`oA++1`oB
PQ
repl_delivery
In
PQ

# Step 4: Dealing with deliveries



# Inventory policy (R,s,S)+b

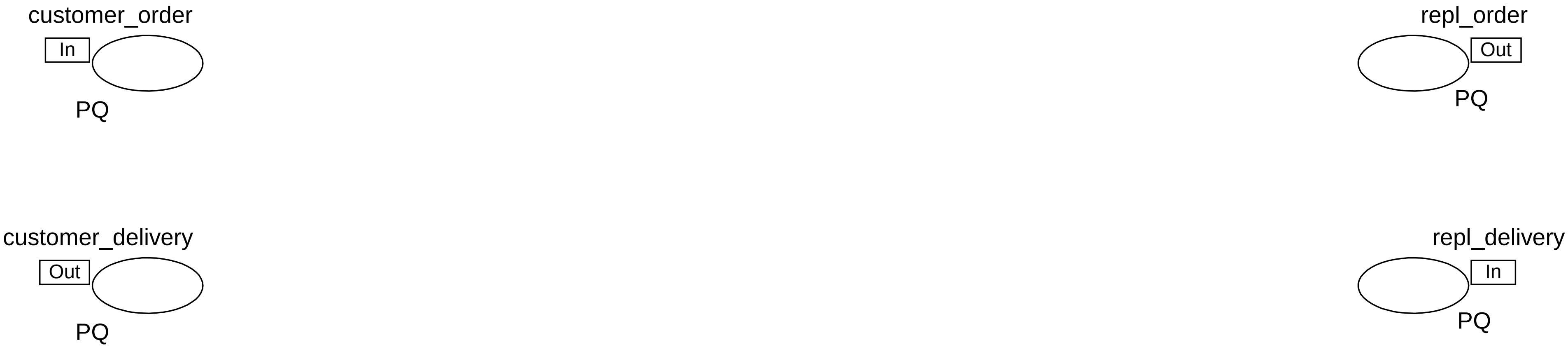
- Complete backordering, i.e., no lost sales.
- Periodic review, i.e., *R* is review period.
- Order point *s.*
- Variable order quantity *S,* i.e., *S* is the order-up-to-level.

Every R time units, the inventory position is compared with s. If it is below s, the inventory position is raised to S by placing an order.

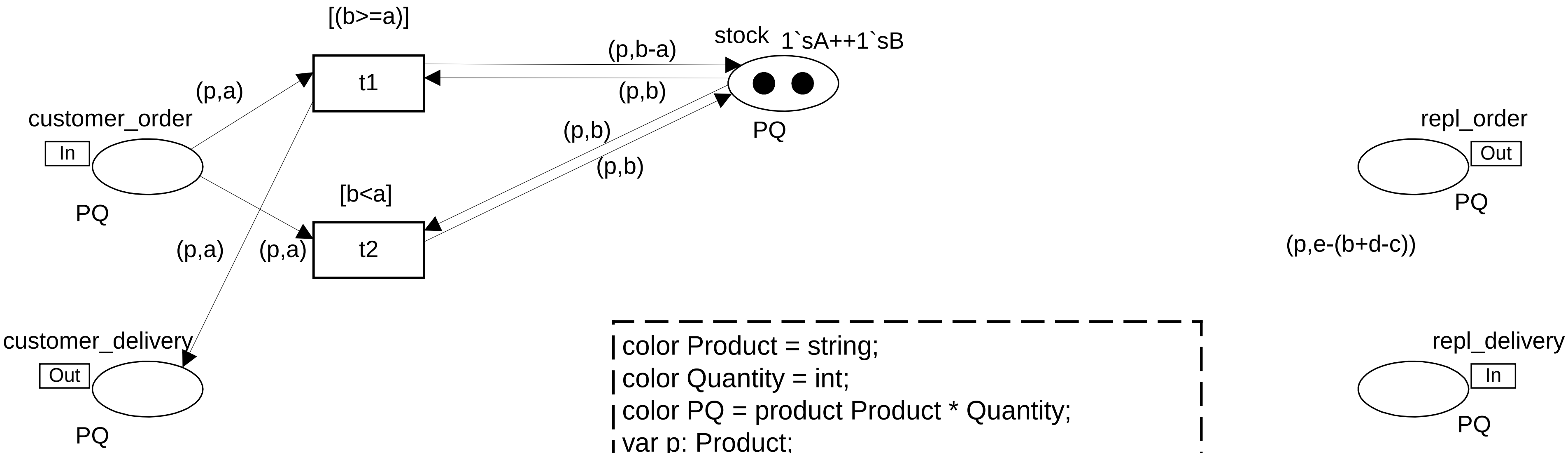
Recall: inventory position = on-hand stock + ordered – back orders.

# Step 1: Port nodes (as before)

```
color Product = string;
color Quantity = int;
color PQ = product Product * Quantity;
```



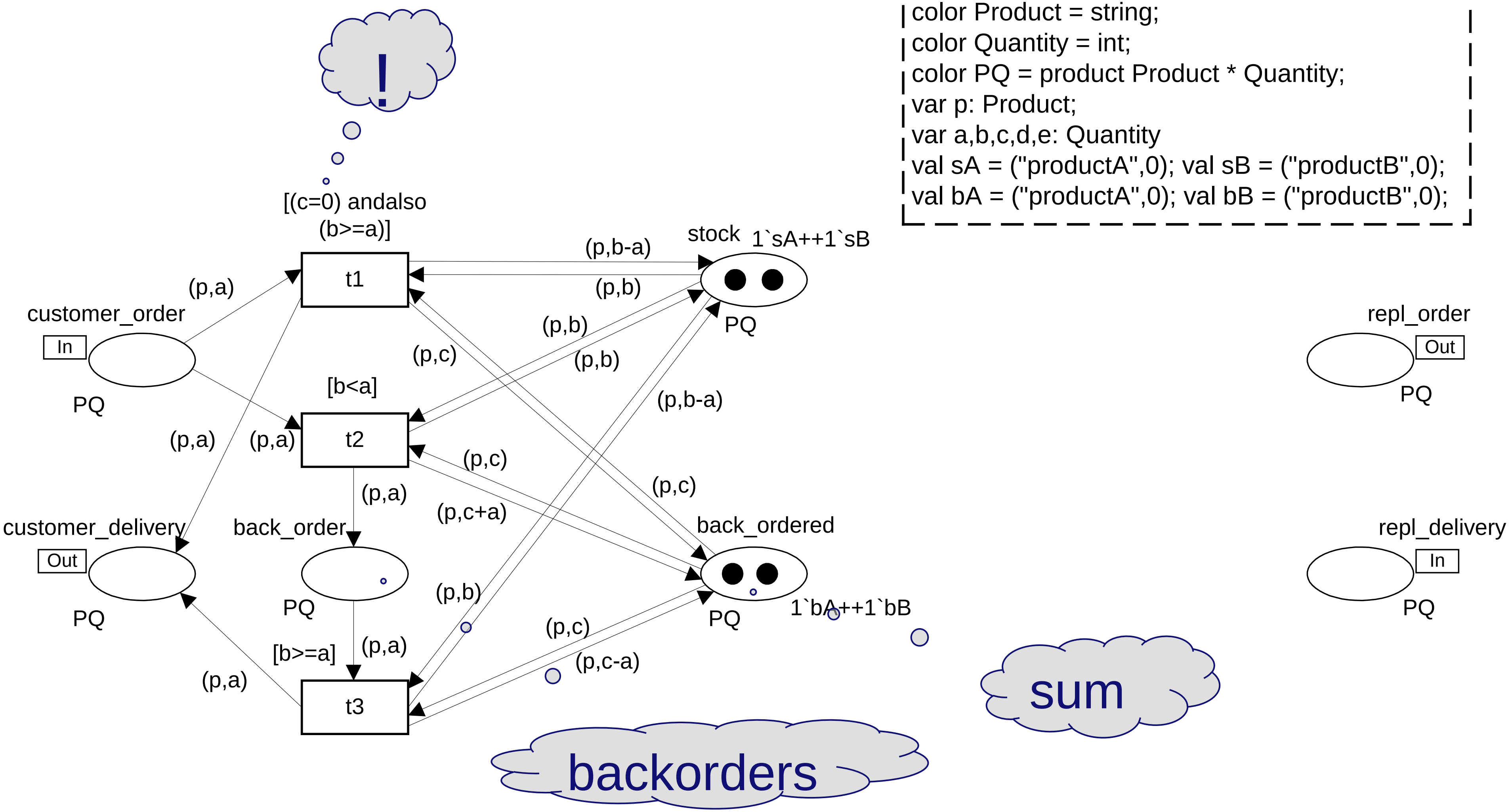
# Step 2: Handle customer order (as before)



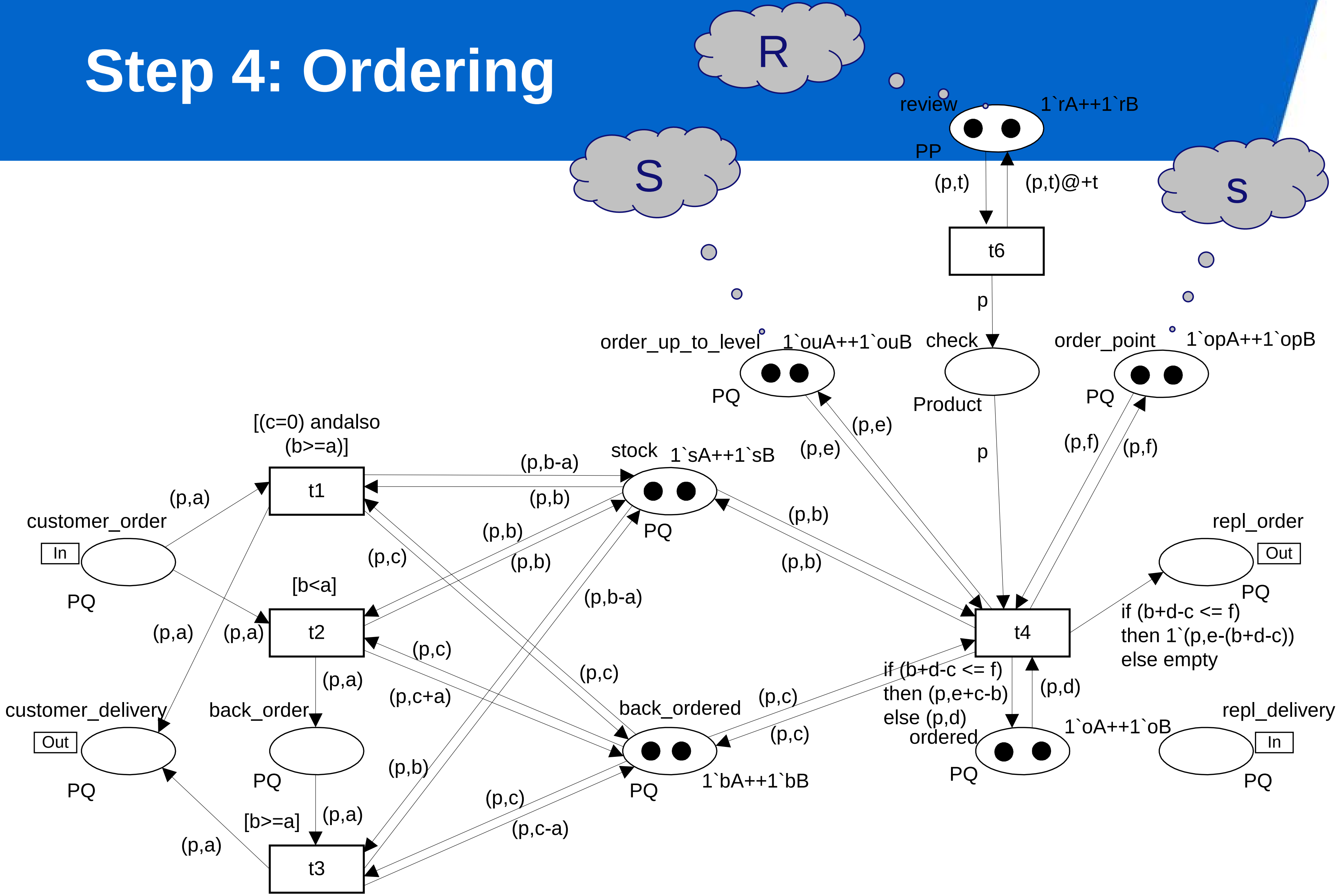
```
color Product = string;
color Quantity = int;
color PQ = product Product * Quantity;
var p: Product;
var a,b,c,d,e: Quantity
val sA = ("productA",0); val sB = ("productB",0);
```

# Step 3: Adding back orders

```
color Product = string;
color Quantity = int;
color PQ = product Product * Quantity;
var p: Product;
var a,b,c,d,e: Quantity
val sA = ("productA",0); val sB = ("productB",0);
val bA = ("productA",0); val bB = ("productB",0);
```



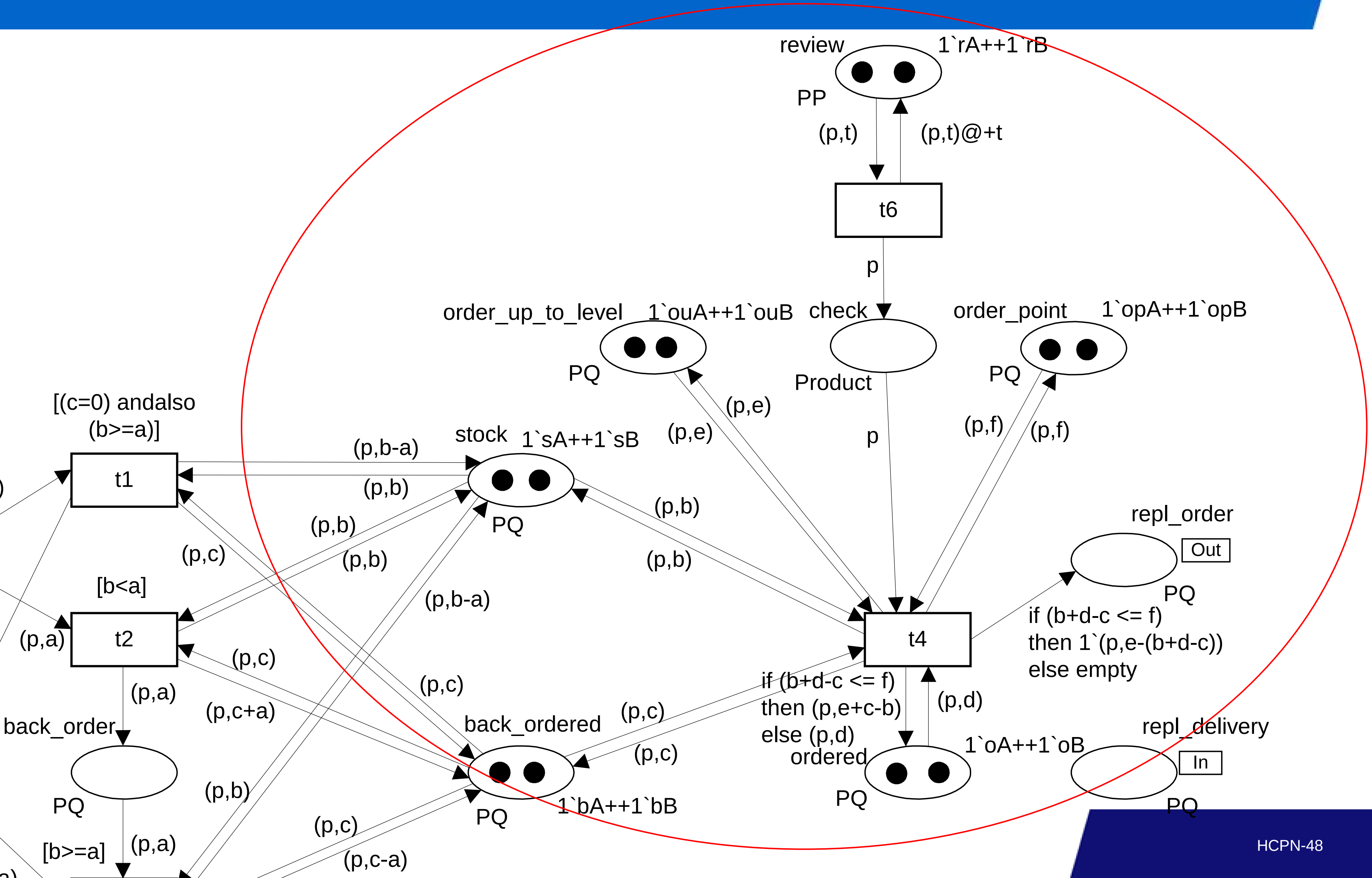
# Step 4: Ordering



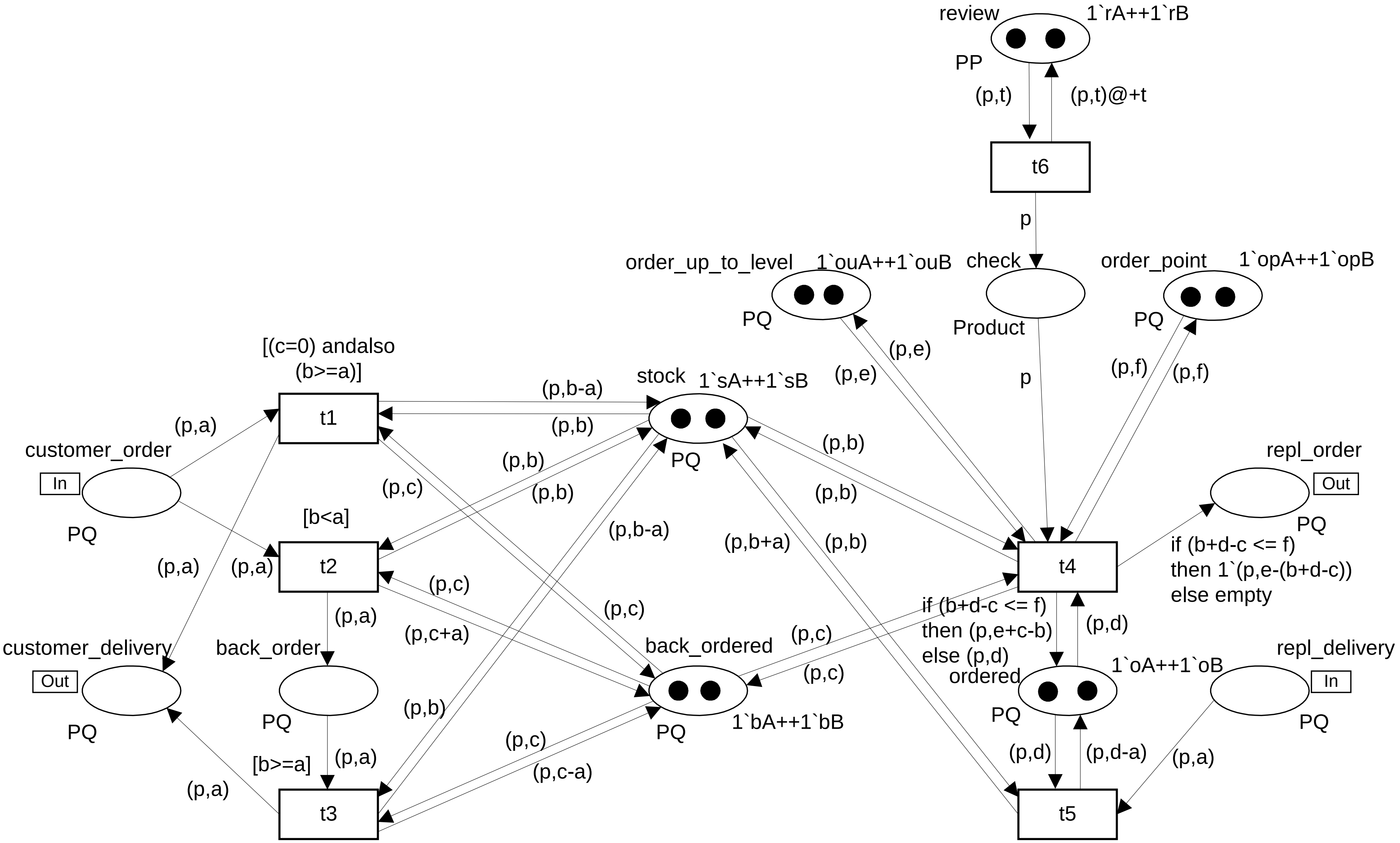
# Declarations

```
color Product = string;
color Quantity = int;
color PQ = product Product * Quantity;
color Period = int;
color PP = product Product * Period timed;
var p: Product;
var t: Period;
var a,b,c,d,e,f: Quantity
val sA = ("productA",0); val sB = ("productB",0);
val oA = ("productA",0); val oB = ("productB",0);
val bA = ("productA",0); val bB = ("productB",0);
val ouA = ("productA",150); val ouB = ("productB",100);
val opA = ("productA",50); val opB = ("productA",60);
val rA = ("productA",3);
val rB = ("productB",4);
```

# Zoom in


review
1`rA++1`rB
PP
(p,t)
(p,t)@+t
t6
p
order_up_to_level
1`ouA++1`ouB
check
order_point
1`opA++1`opB
PQ
Product
PQ
(p,e)
(p,e)
p
(p,f)
(p,f)
[(c=0) andalso
(b>=a)]
(p,b-a)
stock
1`sA++1`sB
t1
(p,b)
(p,b)
PQ
(p,b)
(p,b)
(p,c)
(p,b)
repl_order
Out
PQ
[b<a]
(p,b-a)
if (b+d-c <= f)
then 1`(p,e-(b+d-c))
else empty
(p,a)
t2
t4
(p,c)
(p,c)
(p,a)
if (b+d-c <= f)
then (p,e+c-b)
else (p,d)
(p,d)
(p,c+a)
back_ordered
(p,c)
back_order
repl_delivery
ordered
1`oA++1`oB
In
(p,c)
PQ
(p,b)
PQ
PQ
1`bA++1`bB
PQ
PQ
[b>=a]
(p,a)
(p,c)
(p,c-a)

# Step 6: Dealing with deliveries



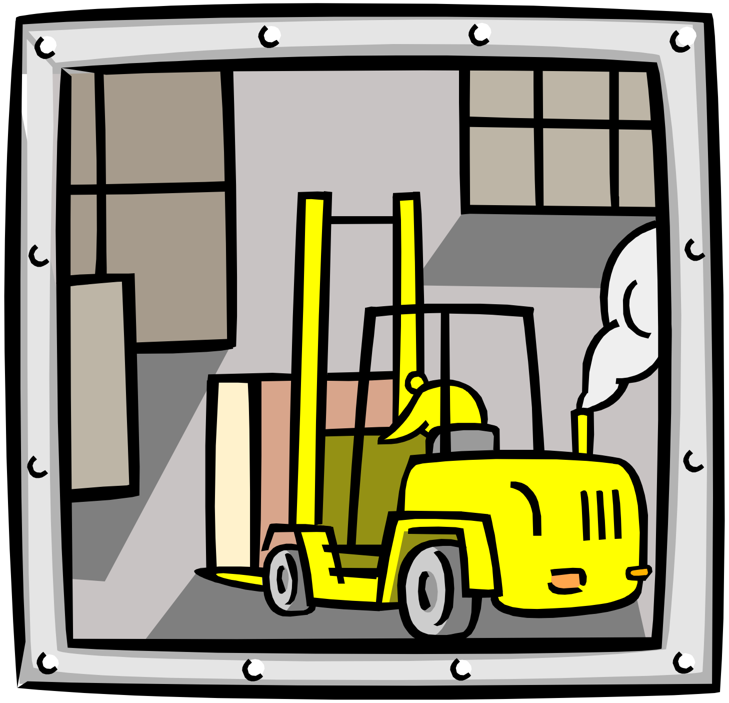
# Six remaining inventory policies

(s,S)-b

(R,s,S)-b

(s,Q)+b

(R,S)-b



(s,Q)-b ☑

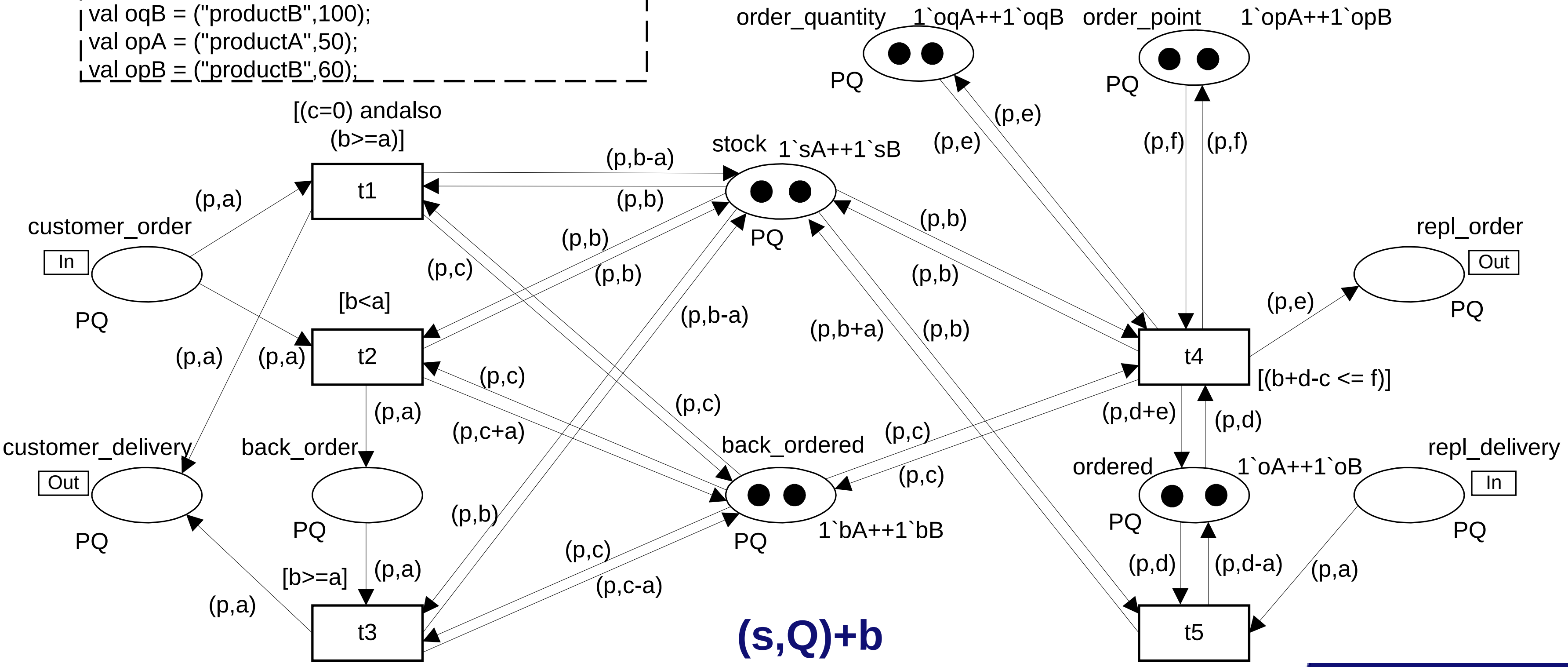
(s,S)+b

(R,s,S)+b ☑

(R,S)+b

# Question: Which inventory policy?

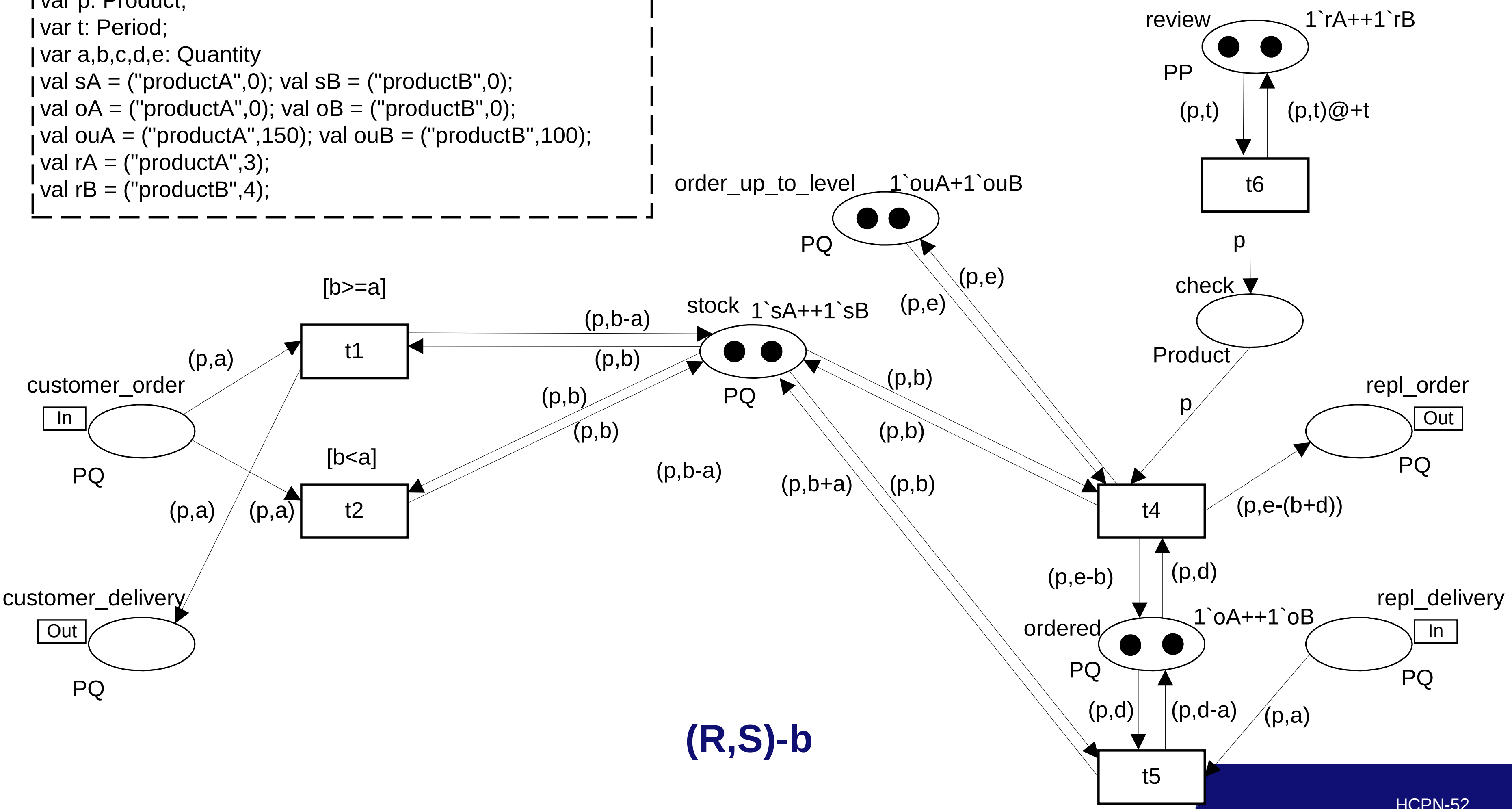
```
color Product = string;
color Quantity = int;
color PQ = product Product * Quantity;
var p: Product;
var a,b,c,d,e,f: Quantity
val sA = ("productA",0);
val sB = ("productB",0);
val oA = ("productA",0);
val oB = ("productB",0);
val bA = ("productA",0);
val bB = ("productB",0);
val oqA = ("productA",150);
val oqB = ("productB",100);
val opA = ("productA",50);
val opB = ("productB",60);
```



**(s,Q)+b**

# Question: Which inventory policy?

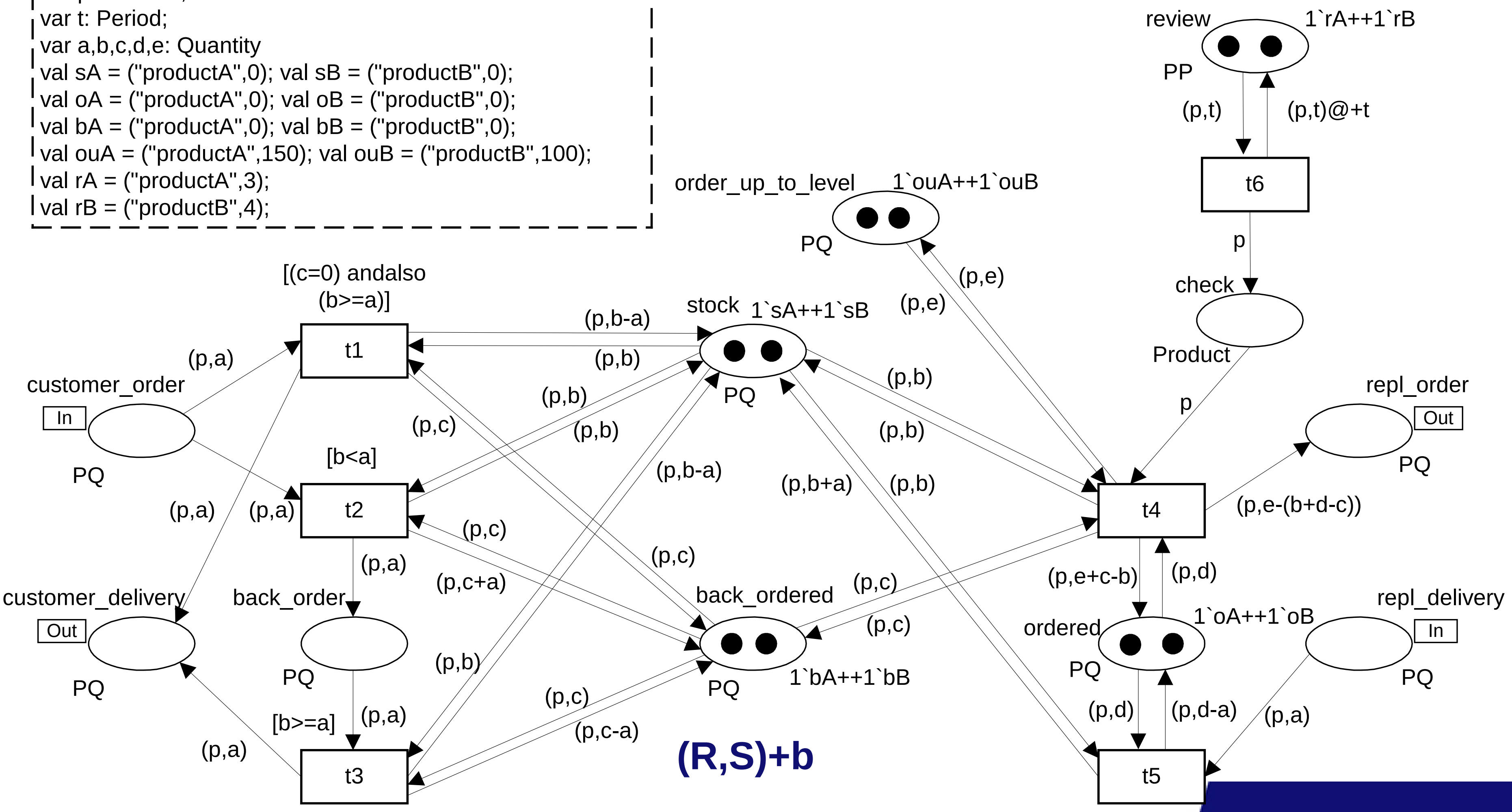
```
color Product = string;
color Quantity = int;
color PQ = product Product * Quantity;
color Period = int;
color PP = product Product * Period timed;
var p: Product;
var t: Period;
var a,b,c,d,e: Quantity
val sA = ("productA",0); val sB = ("productB",0);
val oA = ("productA",0); val oB = ("productB",0);
val ouA = ("productA",150); val ouB = ("productB",100);
val rA = ("productA",3);
val rB = ("productB",4);
```



**(R,S)-b**

# Question: Which inventory policy?

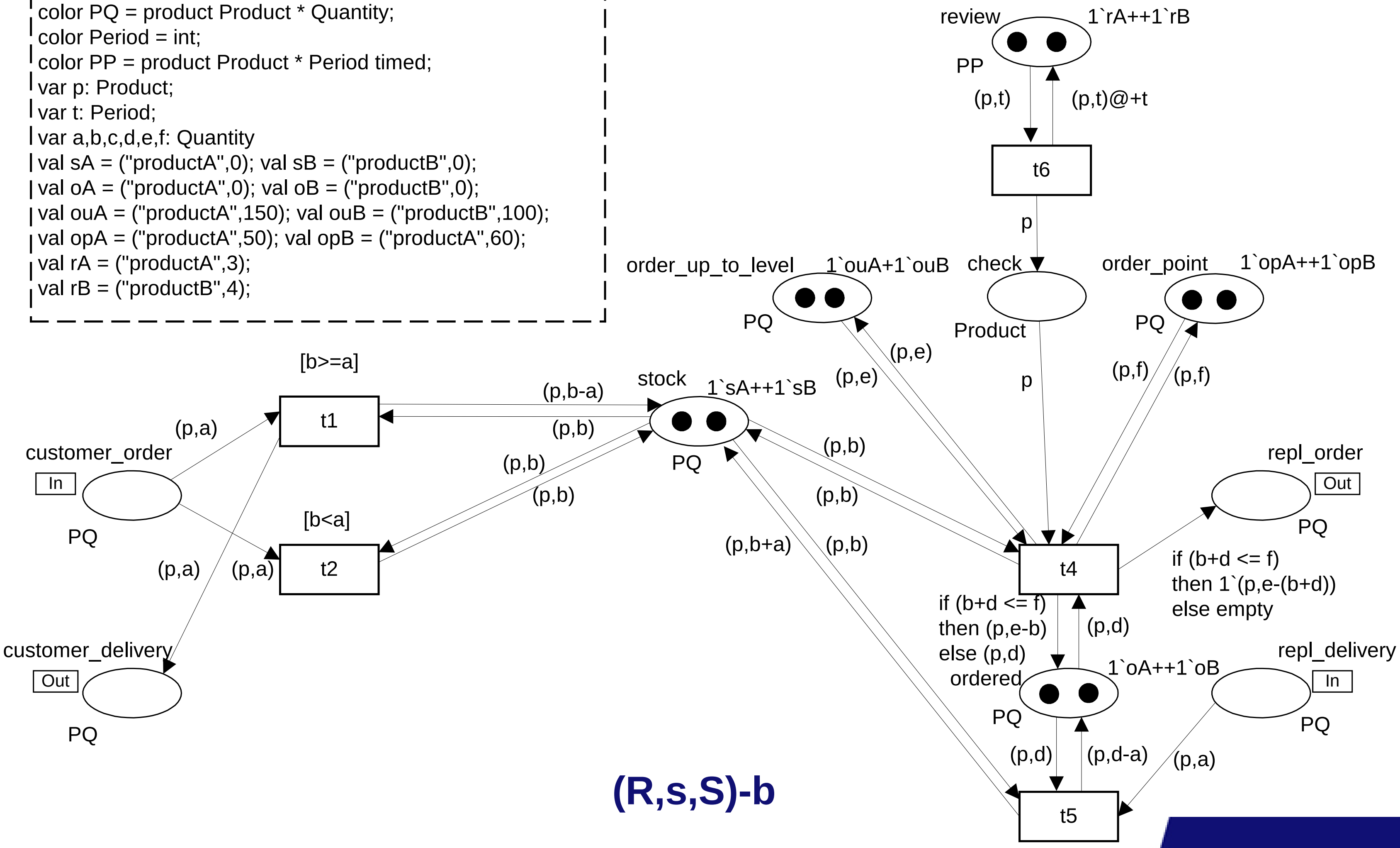
```
color Product = string;
color Quantity = int;
color PQ = product Product * Quantity;
color Period = int;
color PP = product Product * Period timed;
var p: Product;
var t: Period;
var a,b,c,d,e: Quantity
val sA = ("productA",0); val sB = ("productB",0);
val oA = ("productA",0); val oB = ("productB",0);
val bA = ("productA",0); val bB = ("productB",0);
val ouA = ("productA",150); val ouB = ("productB",100);
val rA = ("productA",3);
val rB = ("productB",4);
```



**(R,S)+b**

# Question: Which inventory policy?

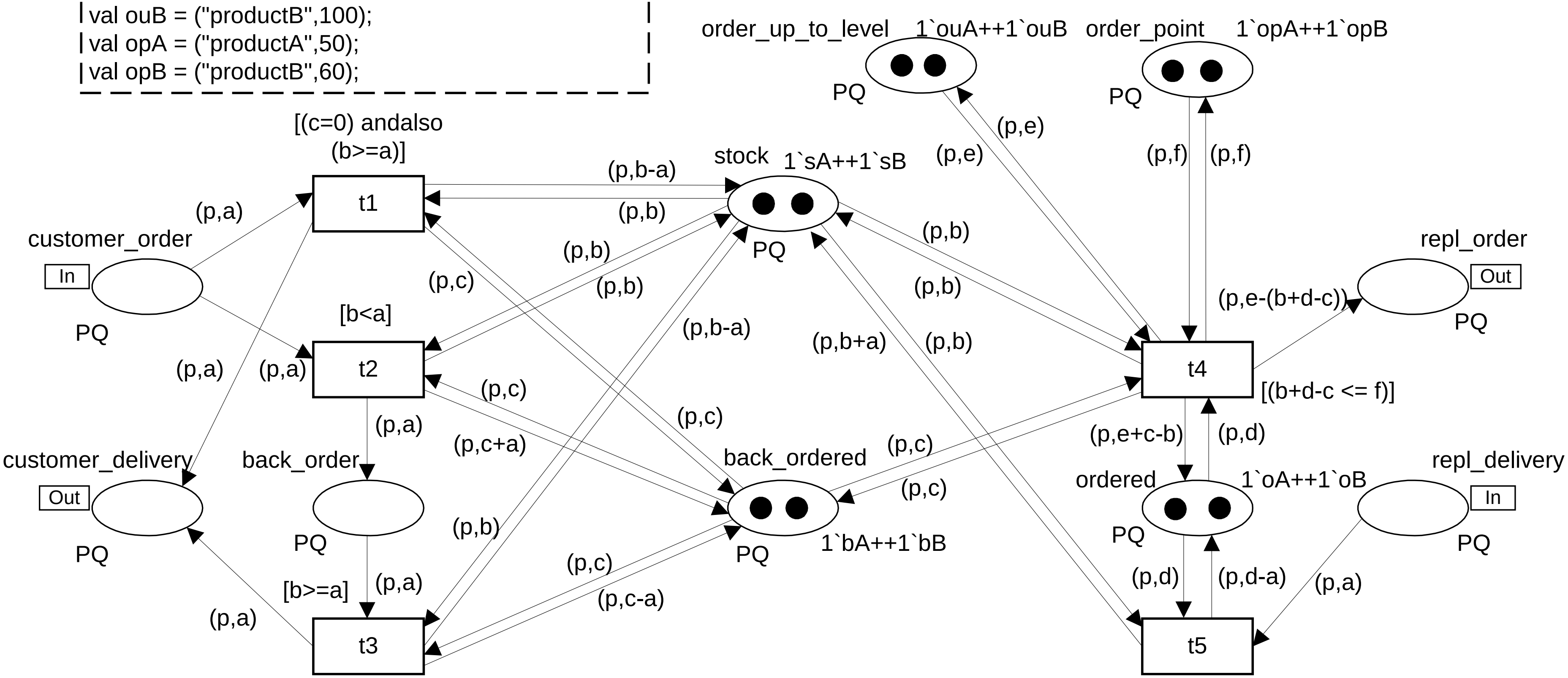
```
color Product = string;
color Quantity = int;
color PQ = product Product * Quantity;
color Period = int;
color PP = product Product * Period timed;
var p: Product;
var t: Period;
var a,b,c,d,e,f: Quantity
val sA = ("productA",0); val sB = ("productB",0);
val oA = ("productA",0); val oB = ("productB",0);
val ouA = ("productA",150); val ouB = ("productB",100);
val opA = ("productA",50); val opB = ("productA",60);
val rA = ("productA",3);
val rB = ("productB",4);
```



**(R,s,S)-b**

# Question: Which inventory policy?

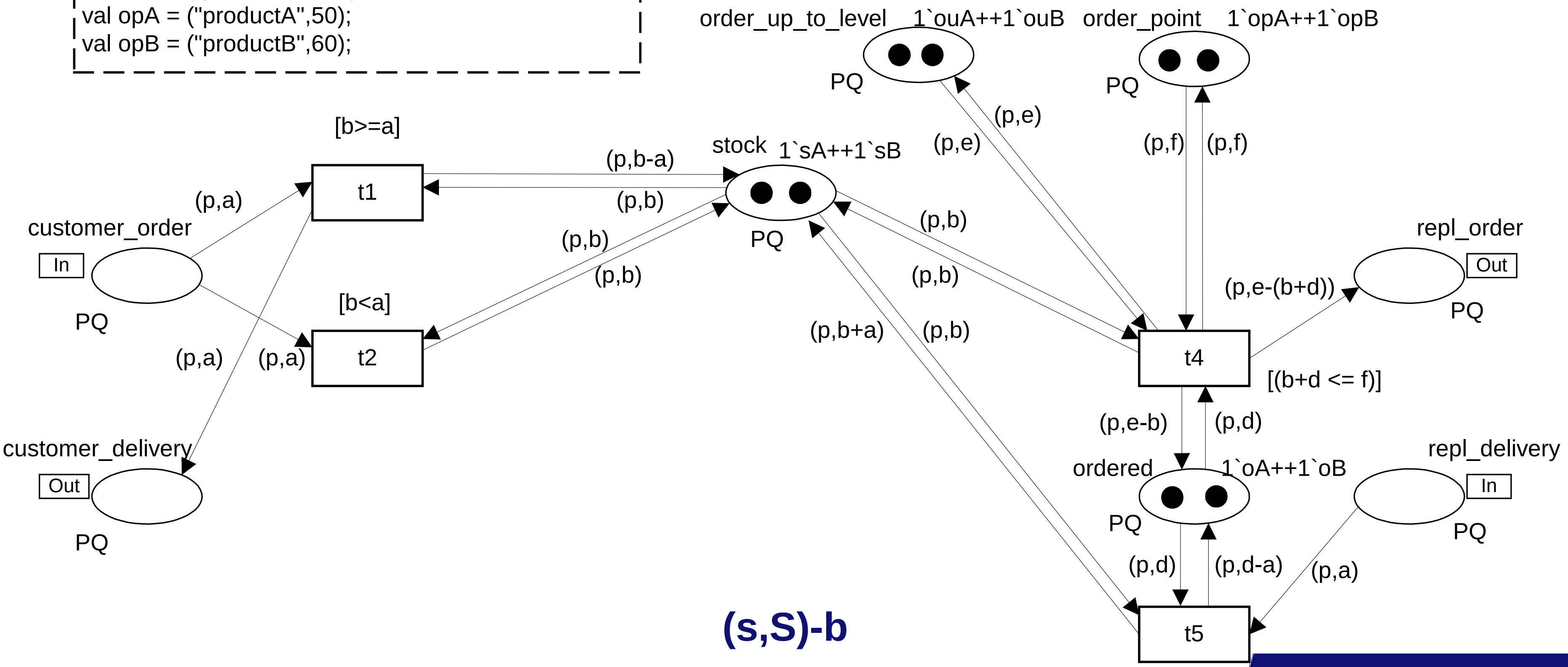
```
color Product = string;
color Quantity = int;
color PQ = product Product * Quantity;
var p: Product;
var a,b,c,d,e,f: Quantity
val sA = ("productA",0);
val sB = ("productB",0);
val oA = ("productA",0);
val oB = ("productB",0);
val bA = ("productA",0);
val bB = ("productB",0);
val ouA = ("productA",150);
val ouB = ("productB",100);
val opA = ("productA",50);
val opB = ("productB",60);
```



(s,S)+b

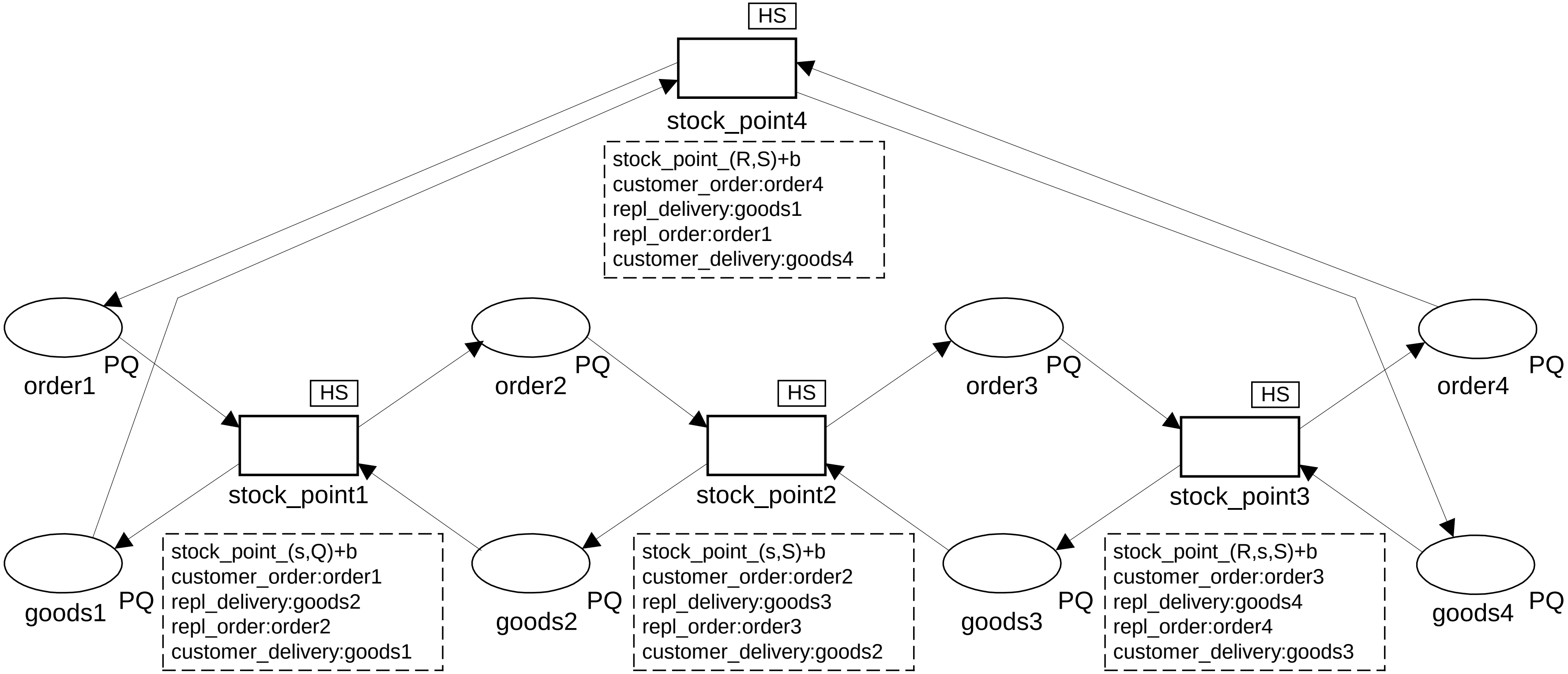
# Question: Which inventory policy?

```
color Product = string;
color Quantity = int;
color PQ = product Product * Quantity;
var p: Product;
var a,b,c,d,e,f: Quantity
val sA = ("productA",0);
val sB = ("productB",0);
val oA = ("productA",0);
val oB = ("productB",0);
val ouA = ("productA",150);
val ouB = ("productB",100);
val opA = ("productA",50);
val opB = ("productB",60);
```



**(s,S)-b**

# Modeling a supply chain



stock_point_(R,S)+b
customer_order:order4
repl_delivery:goods1
repl_order:order1
customer_delivery:goods4

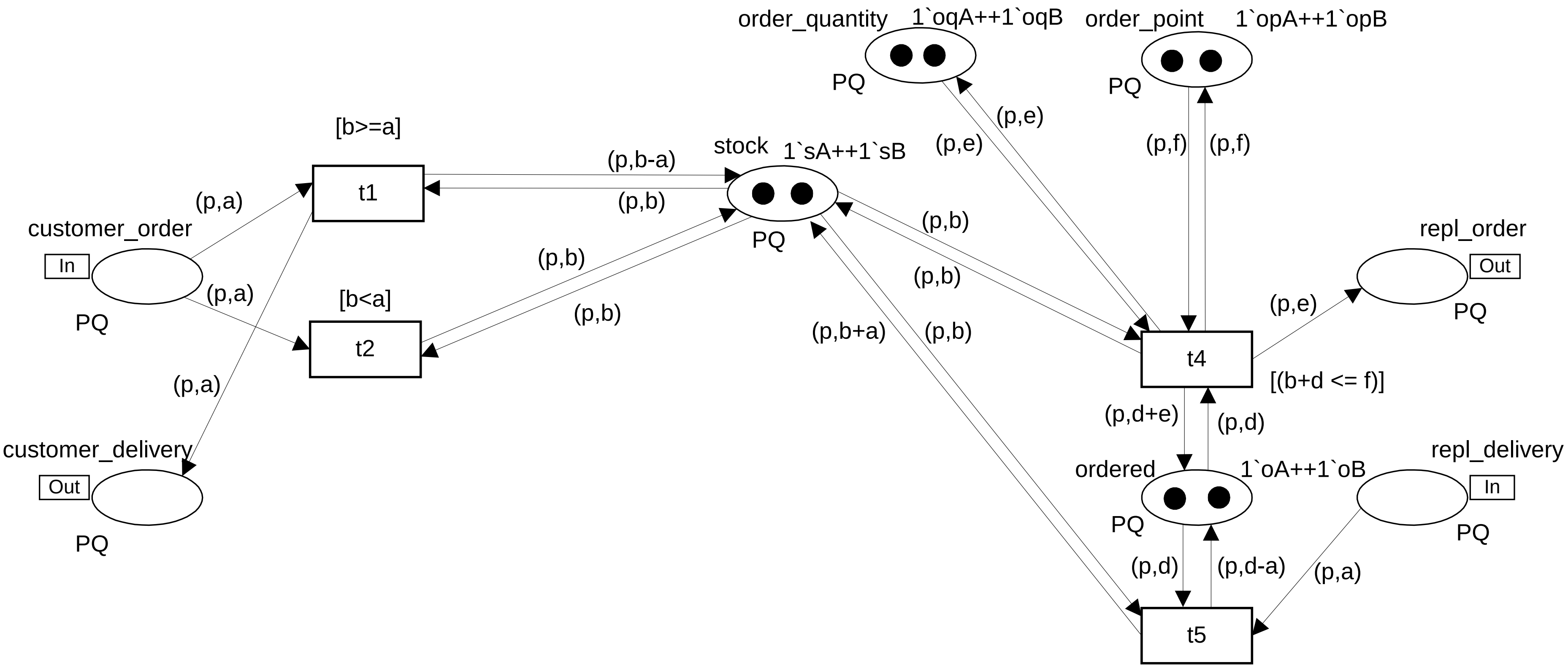
stock_point_(s,Q)+b
customer_order:order1
repl_delivery:goods2
repl_order:order2
customer_delivery:goods1

stock_point_(s,S)+b
customer_order:order2
repl_delivery:goods3
repl_order:order3
customer_delivery:goods2

stock_point_(R,s,S)+b
customer_order:order3
repl_delivery:goods4
repl_order:order4
customer_delivery:goods3

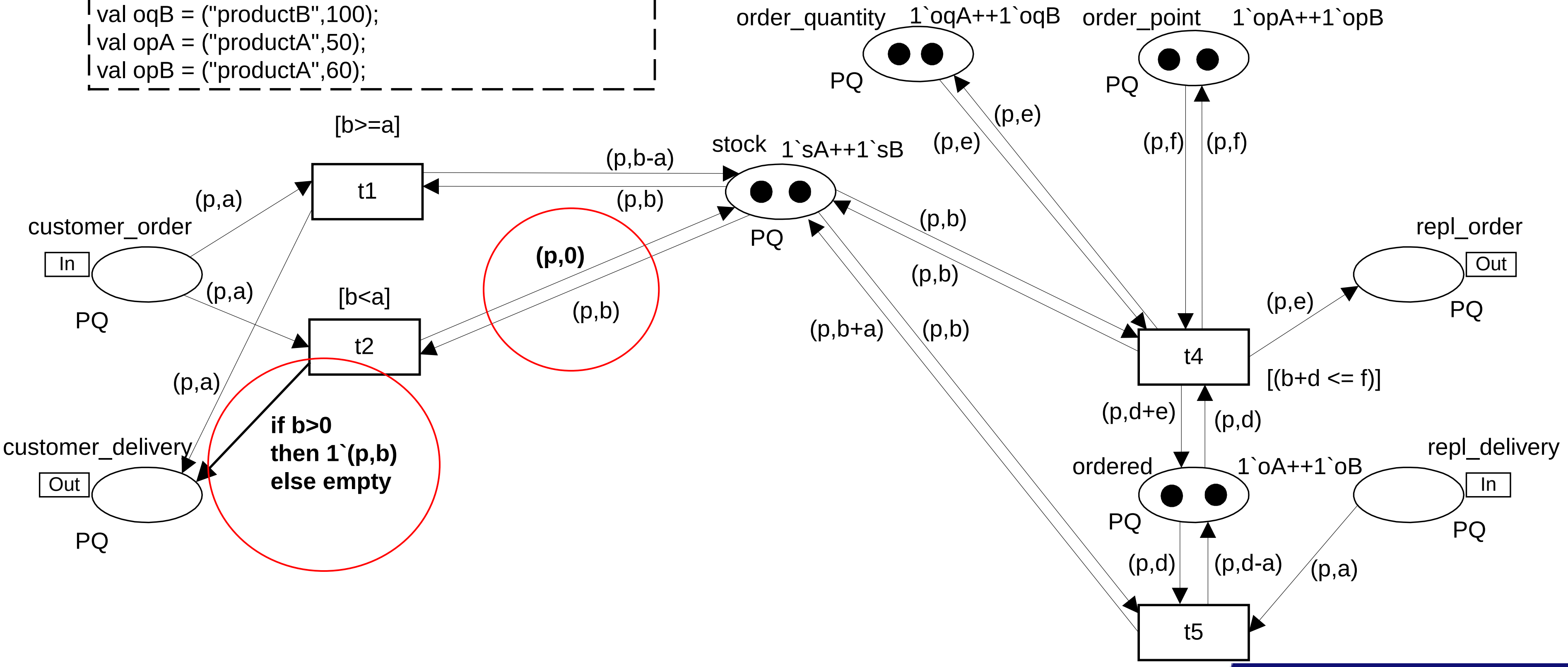
(For simplicity we do not add external configuration places. Direction and circular structure may be confusing.)

# Exercise: Modify to allow for partial shipments



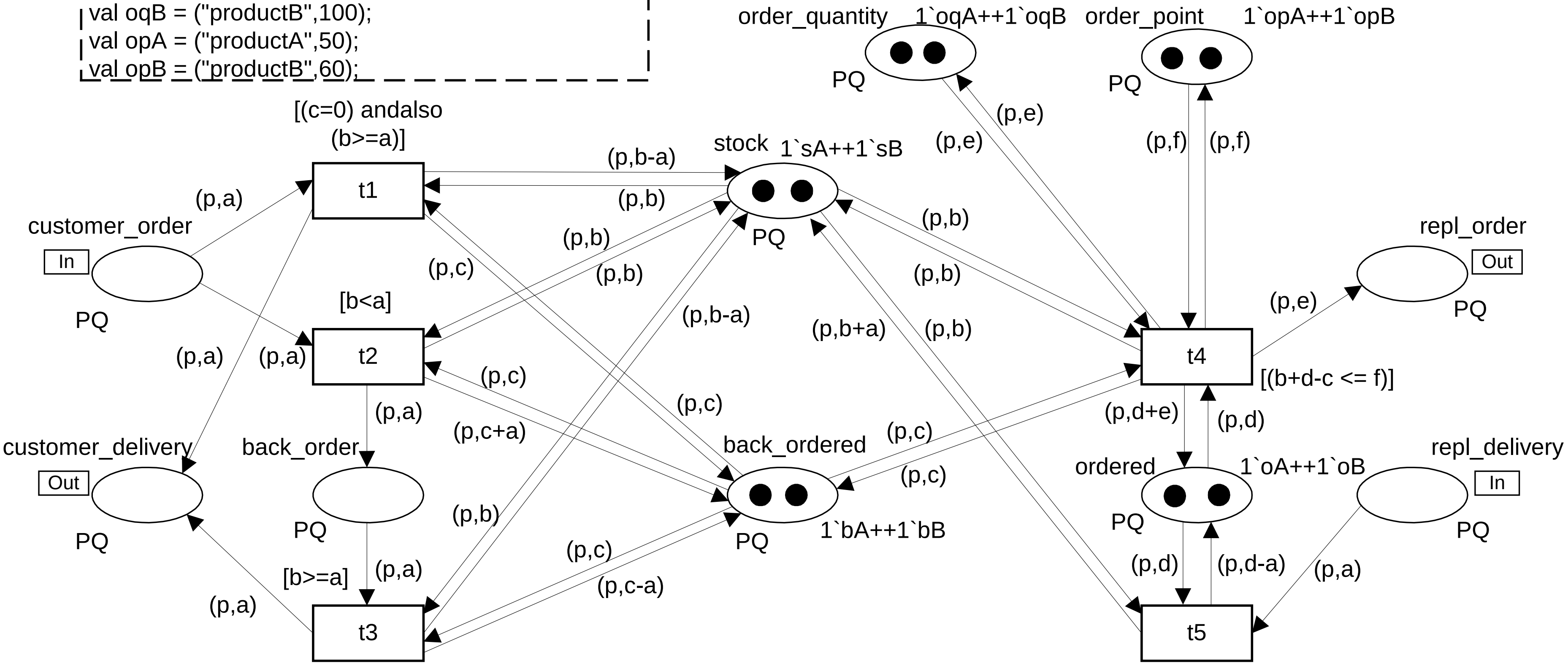
# Solution

```
color Product = string;
color Quantity = int;
color PQ = product Product * Quantity;
var p: Product;
var a,b,c,d,e,f: Quantity
val sA = ("productA",0);
val sB = ("productB",0);
val oA = ("productA",0);
val oB = ("productB",0);
val oqA = ("productA",150);
val oqB = ("productB",100);
val opA = ("productA",50);
val opB = ("productA",60);
```



# Exercise: Modify to allow for partial shipments

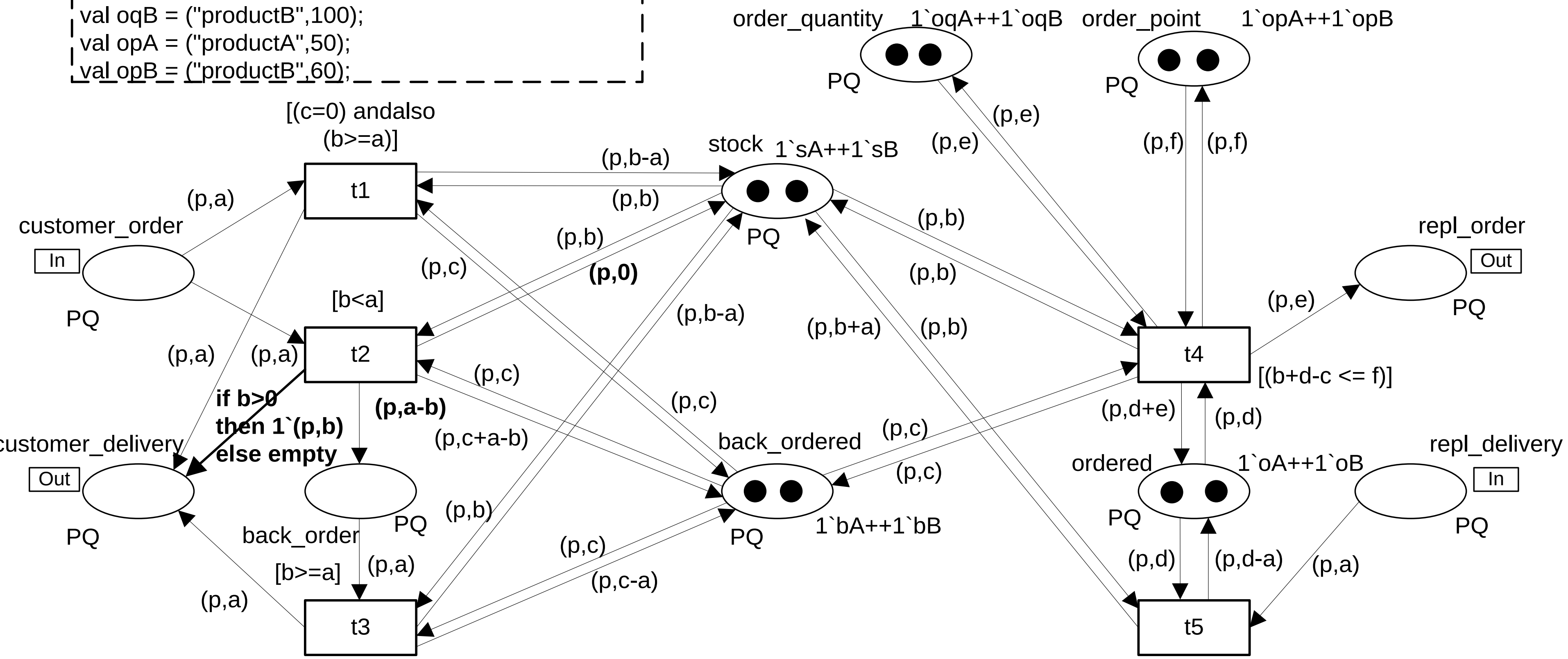
```
color Product = string;
color Quantity = int;
color PQ = product Product * Quantity;
var p: Product;
var a,b,c,d,e,f: Quantity
val sA = ("productA",0);
val sB = ("productB",0);
val oA = ("productA",0);
val oB = ("productB",0);
val bA = ("productA",0);
val bB = ("productB",0);
val oqA = ("productA",150);
val oqB = ("productB",100);
val opA = ("productA",50);
val opB = ("productB",60);
```



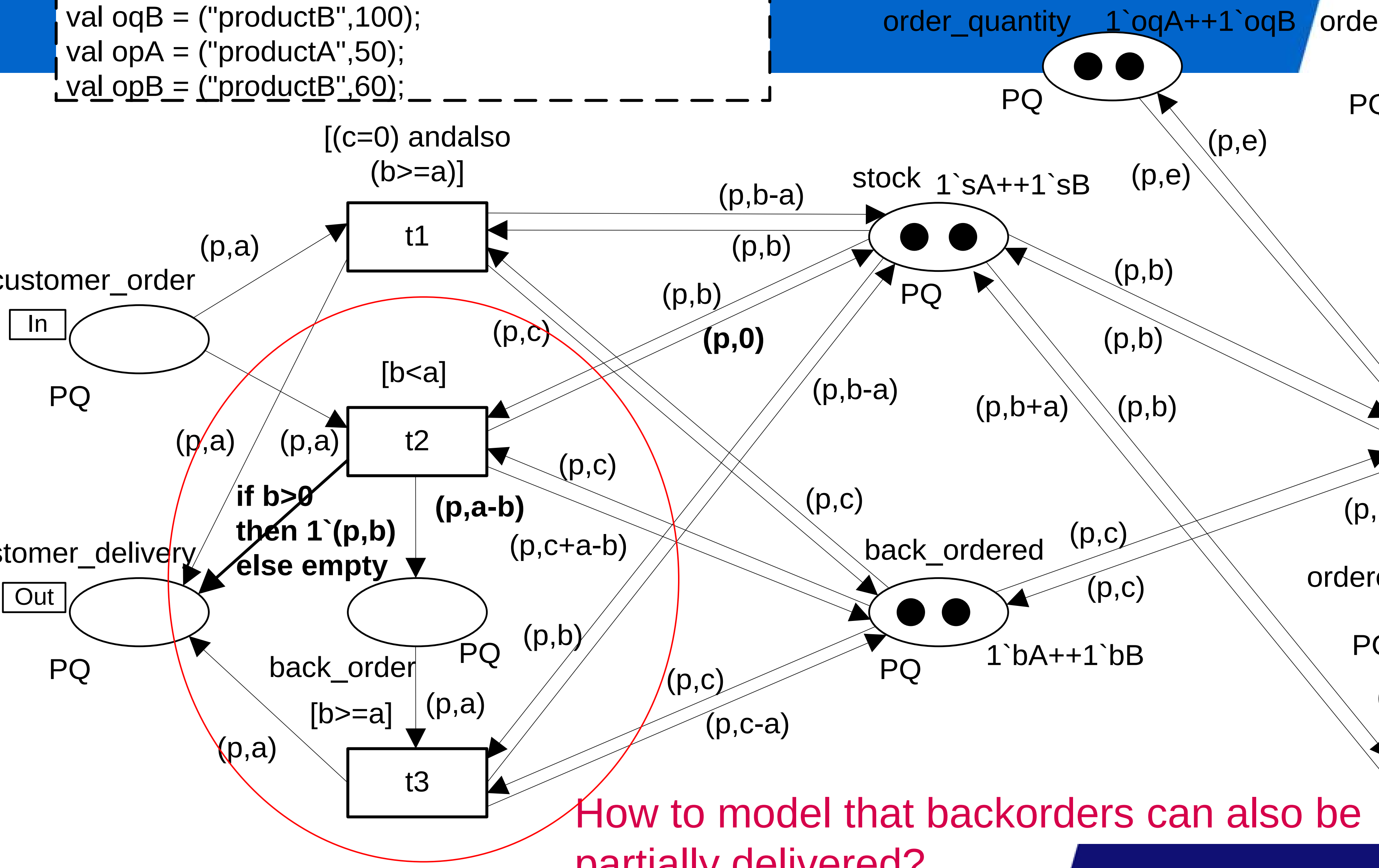
ekordered.)

# Solution

```
color Product = string;
color Quantity = int;
color PQ = product Product * Quantity;
var p: Product;
var a,b,c,d,e,f: Quantity
val sA = ("productA",0);
val sB = ("productB",0);
val oA = ("productA",0);
val oB = ("productB",0);
val bA = ("productA",0);
val bB = ("productB",0);
val oqA = ("productA",150);
val oqB = ("productB",100);
val opA = ("productA",50);
val opB = ("productB",60);
```



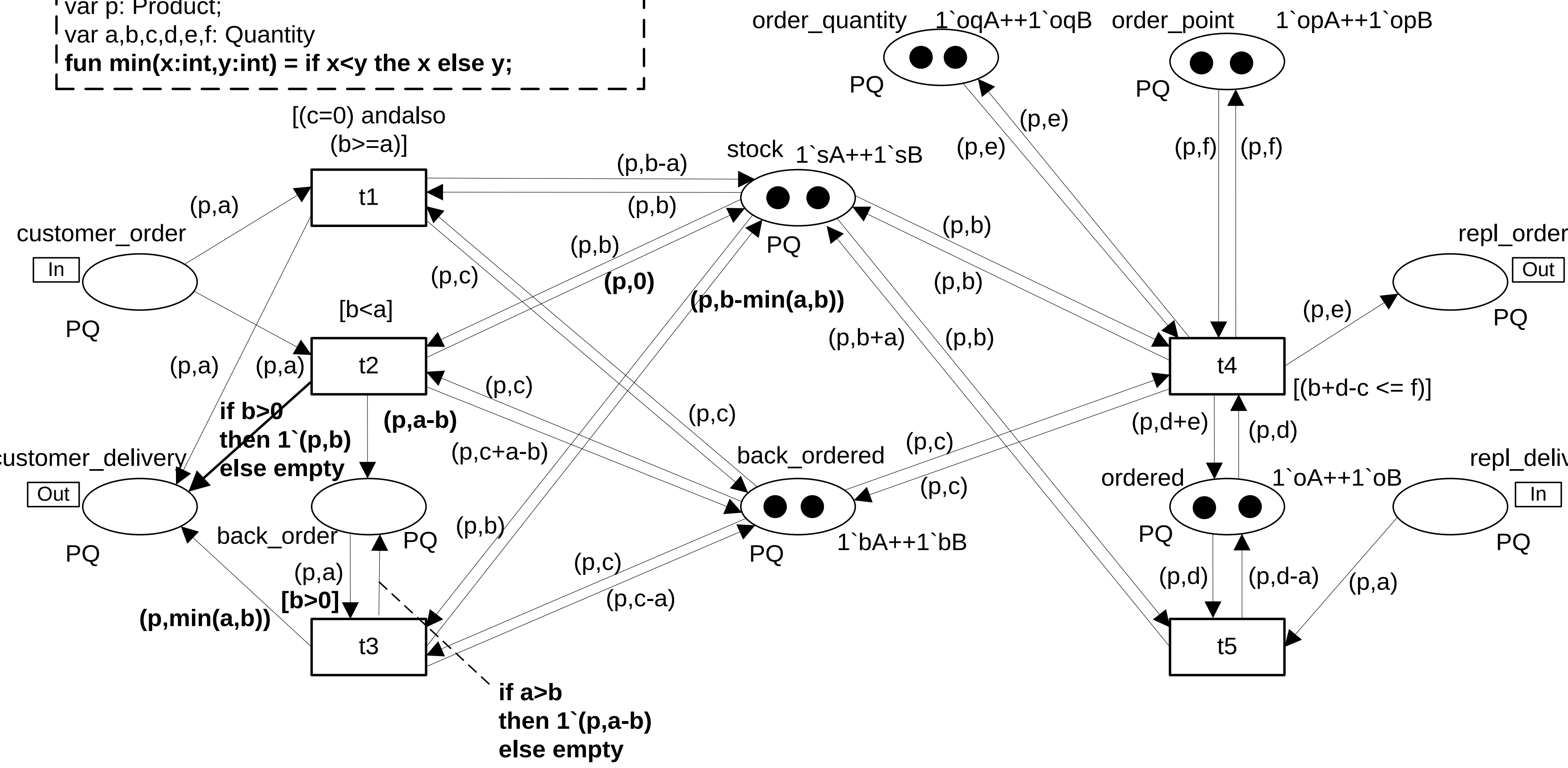
```
val oB = ("productB",0);
val bA = ("productA",0);
val bB = ("productB",0);
val oqA = ("productA",150);
val oqB = ("productB",100);
val opA = ("productA",50);
val opB = ("productB",60);
```



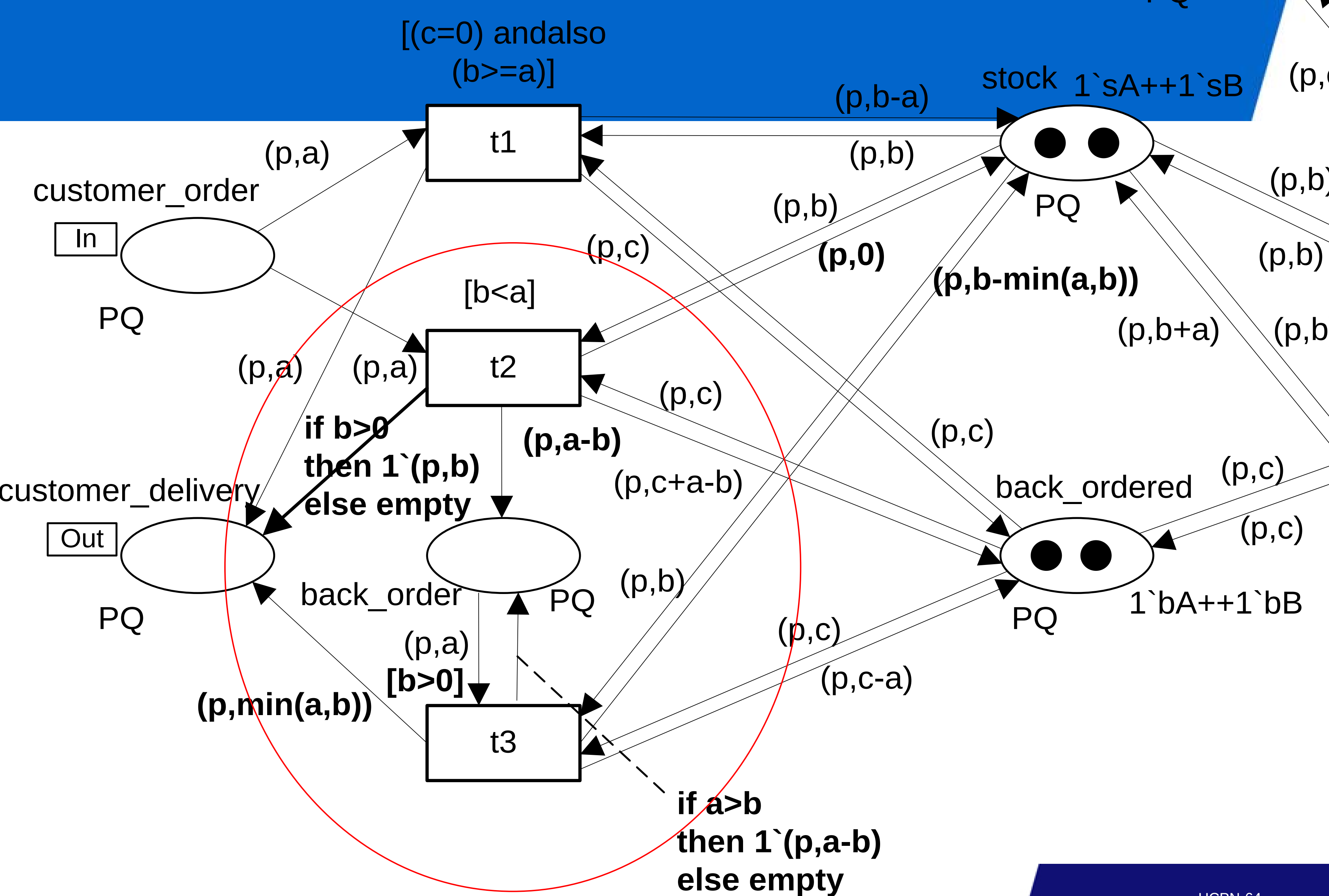
How to model that backorders can also be partially delivered?

# Solution

```
color Product = string;
color Quantity = int;
color PQ = product Product * Quantity;
var p: Product;
var a,b,c,d,e,f: Quantity
```
**fun min(x:int,y:int) = if x<y the x else y;**



**fun min(x:int,y:int) = if x<y the x else y;**



# After studying Chapter 6 one should be able to:

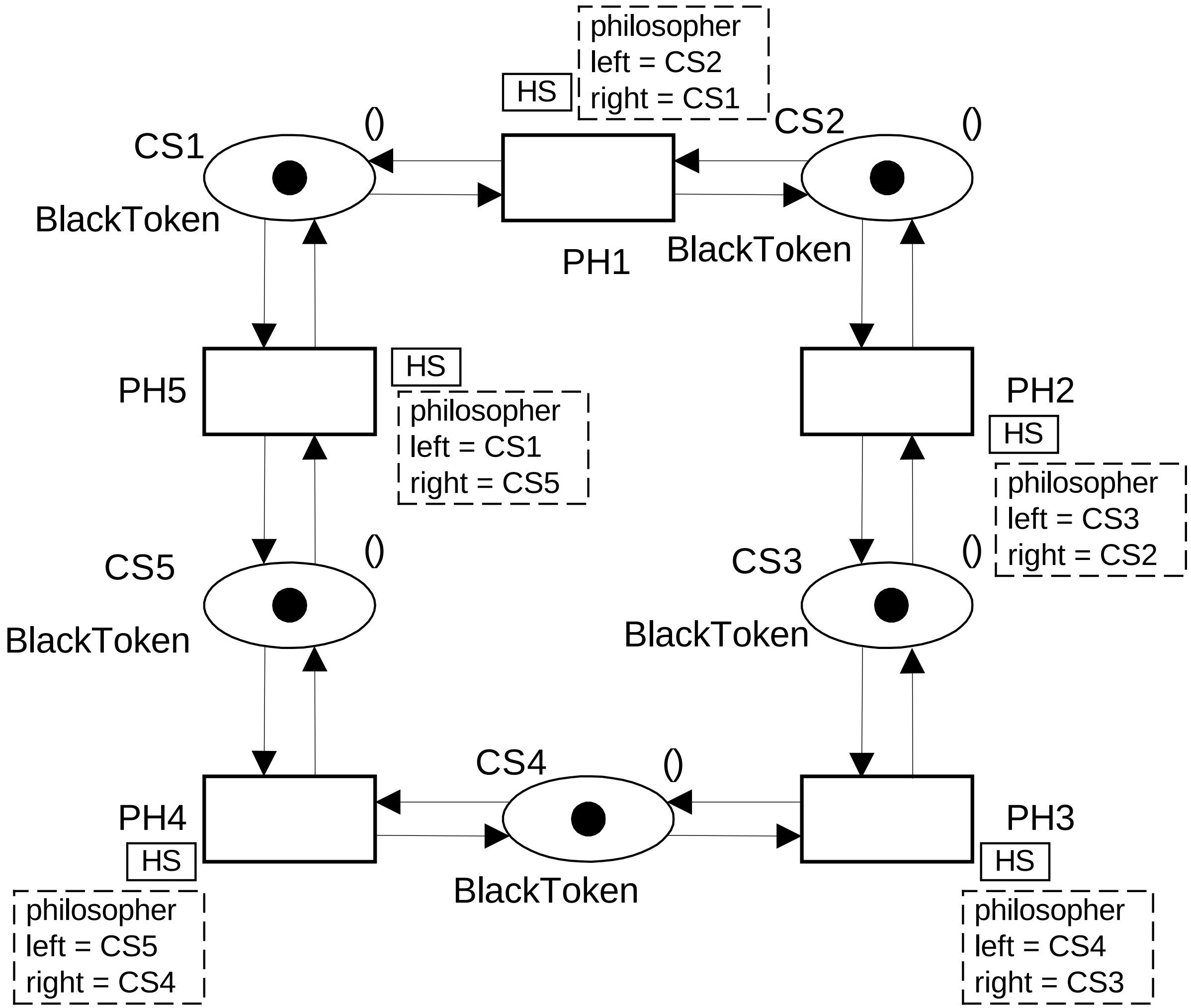
- Flatten a hierarchical CPN model.
- Design a hierarchical CPN model for scratch.
- Modify a hierarchical CPN model.

# Exercise: Five Chinese philosophers

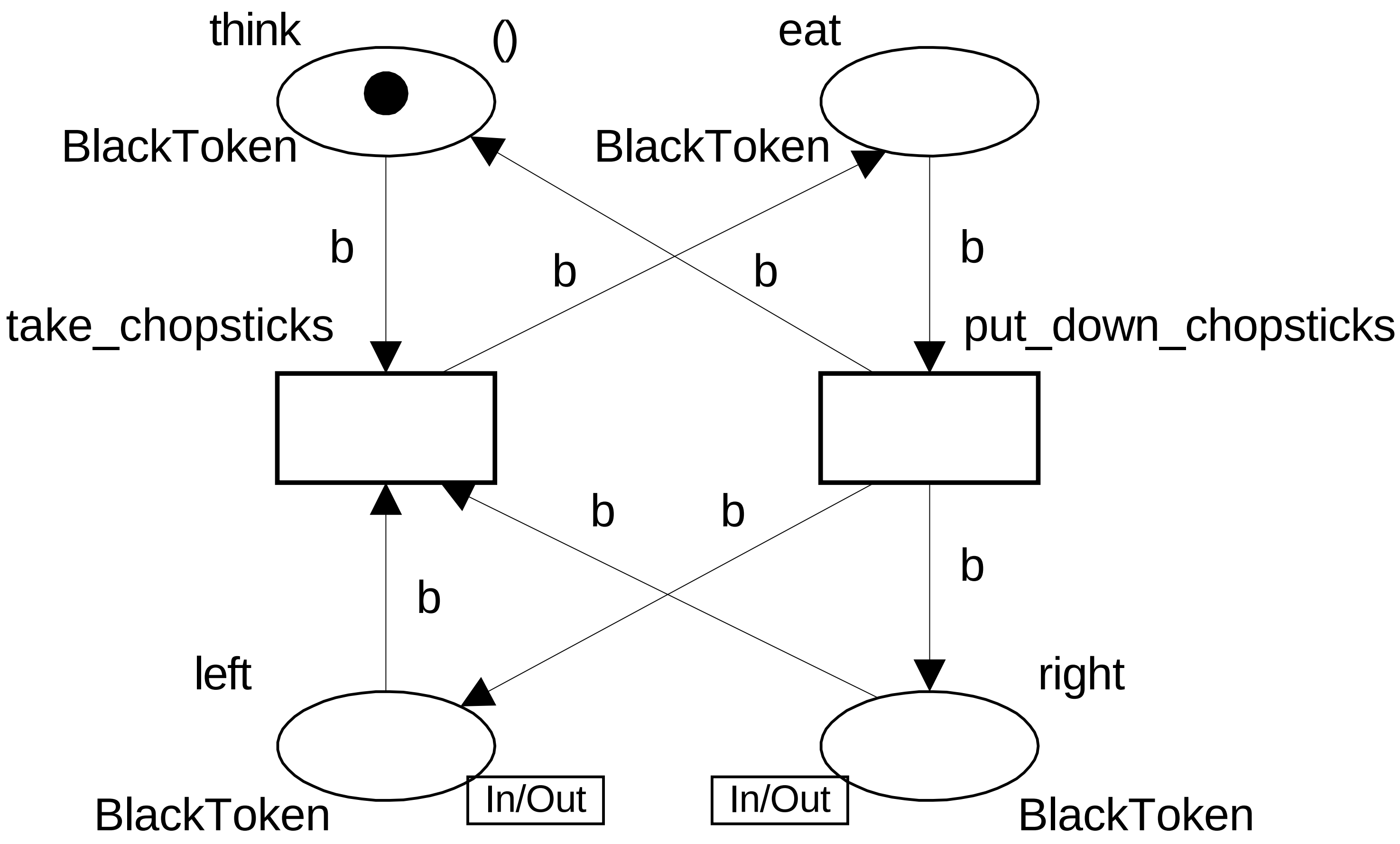
- Make a hierarchical CPN model of five Chinese philosophers alternating between states thinking and eating. To eat two chopsticks are needed. In total there are five chopsticks. The philosophers are sitting in a circle, and need to complete for chopsticks with their direct neighbors (left and right). Assume that both chopsticks need to be taken at the same time.
  Model this using a hierarchical CPN model. Make sure to model the behavior of a philosopher only once and just use the color set BlackToken of type unit.
- Change the model such that philosophers can take one chopstick at a time but avoid deadlocks and a fixed ordering of philosophers.
- Flatten the hierarchical CPN model.

# Solution: Top-level page

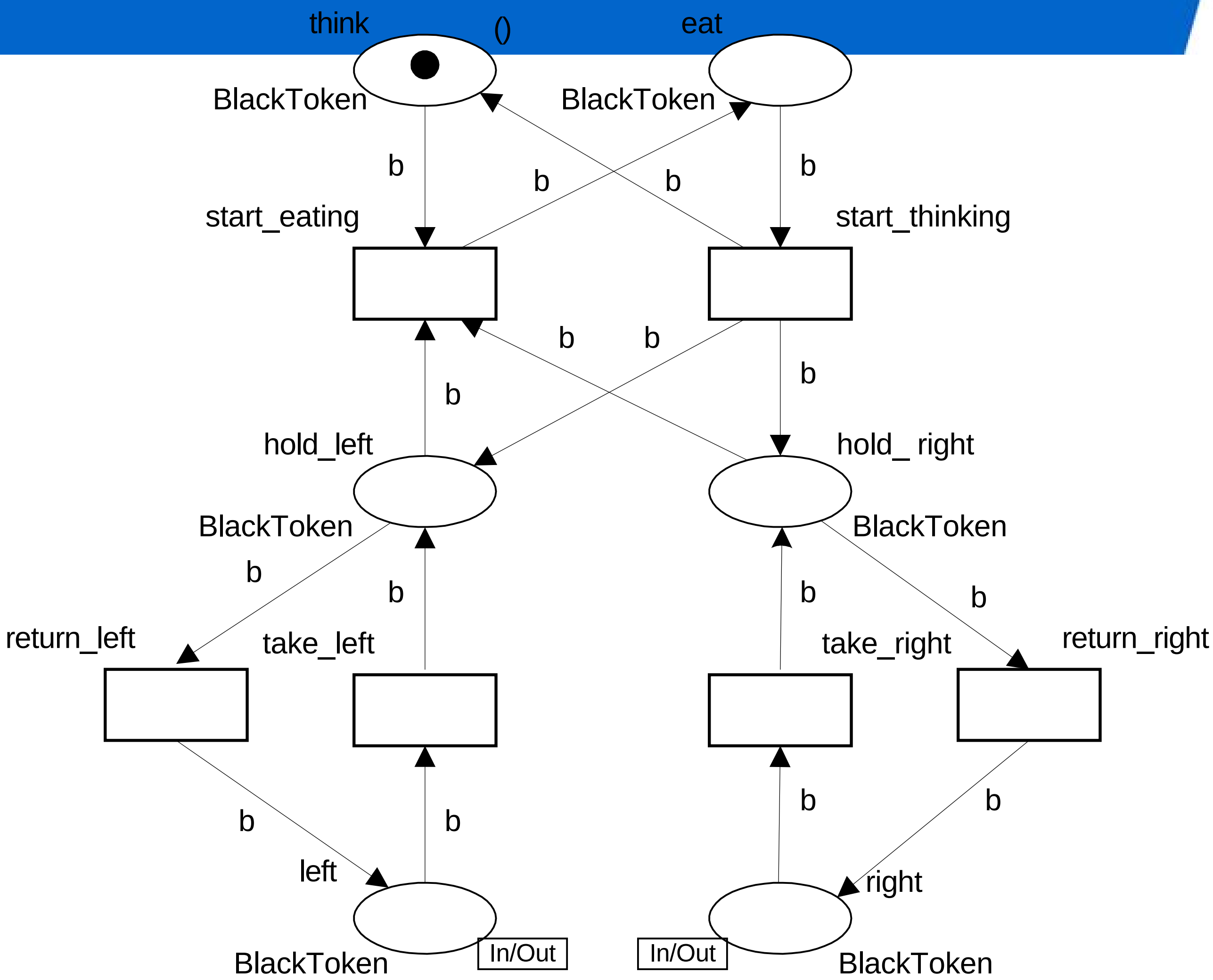
```
color BlackToken = unit;
var b:BackToken
```



# Page philosopher

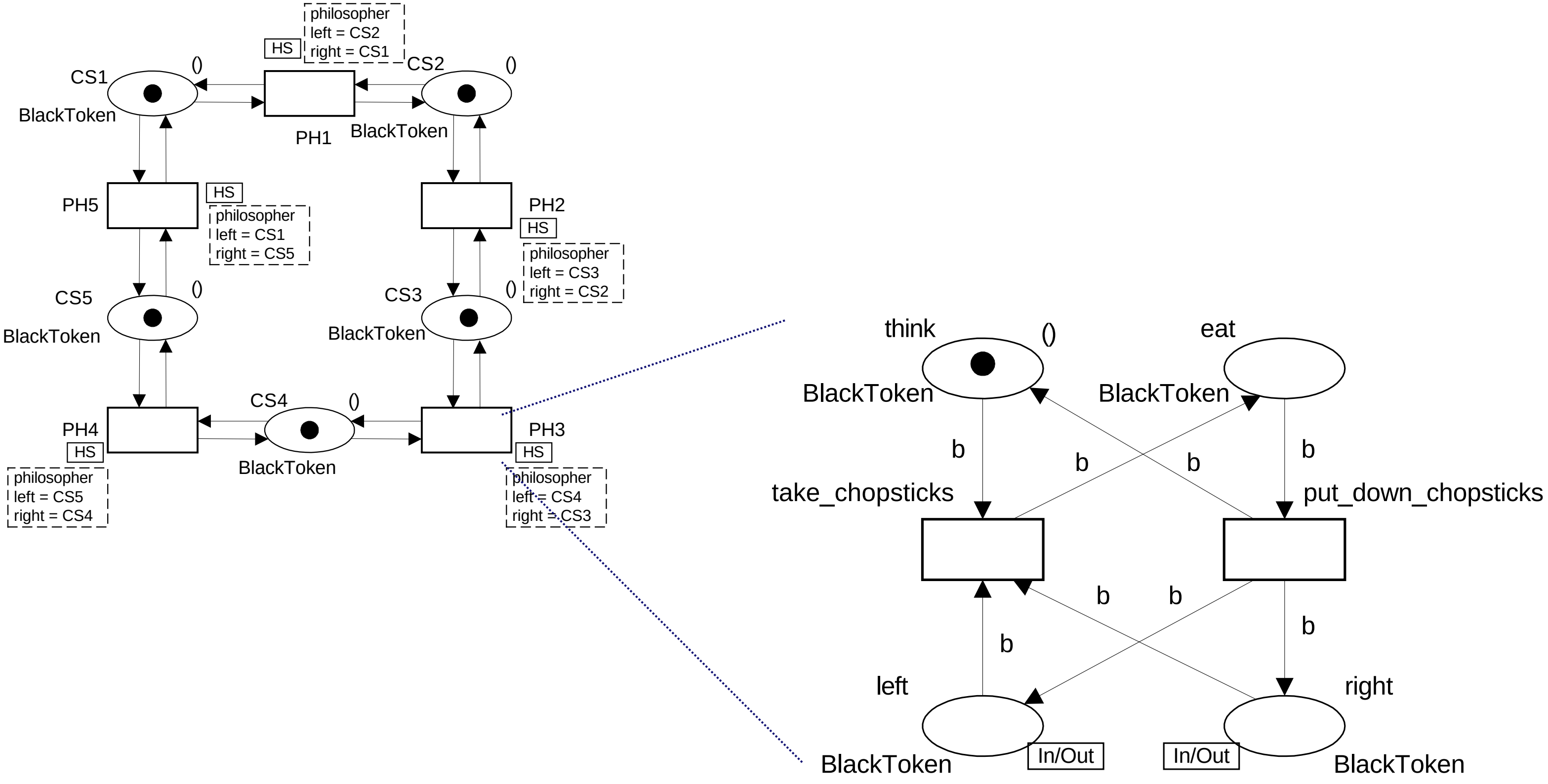


# Alternative page



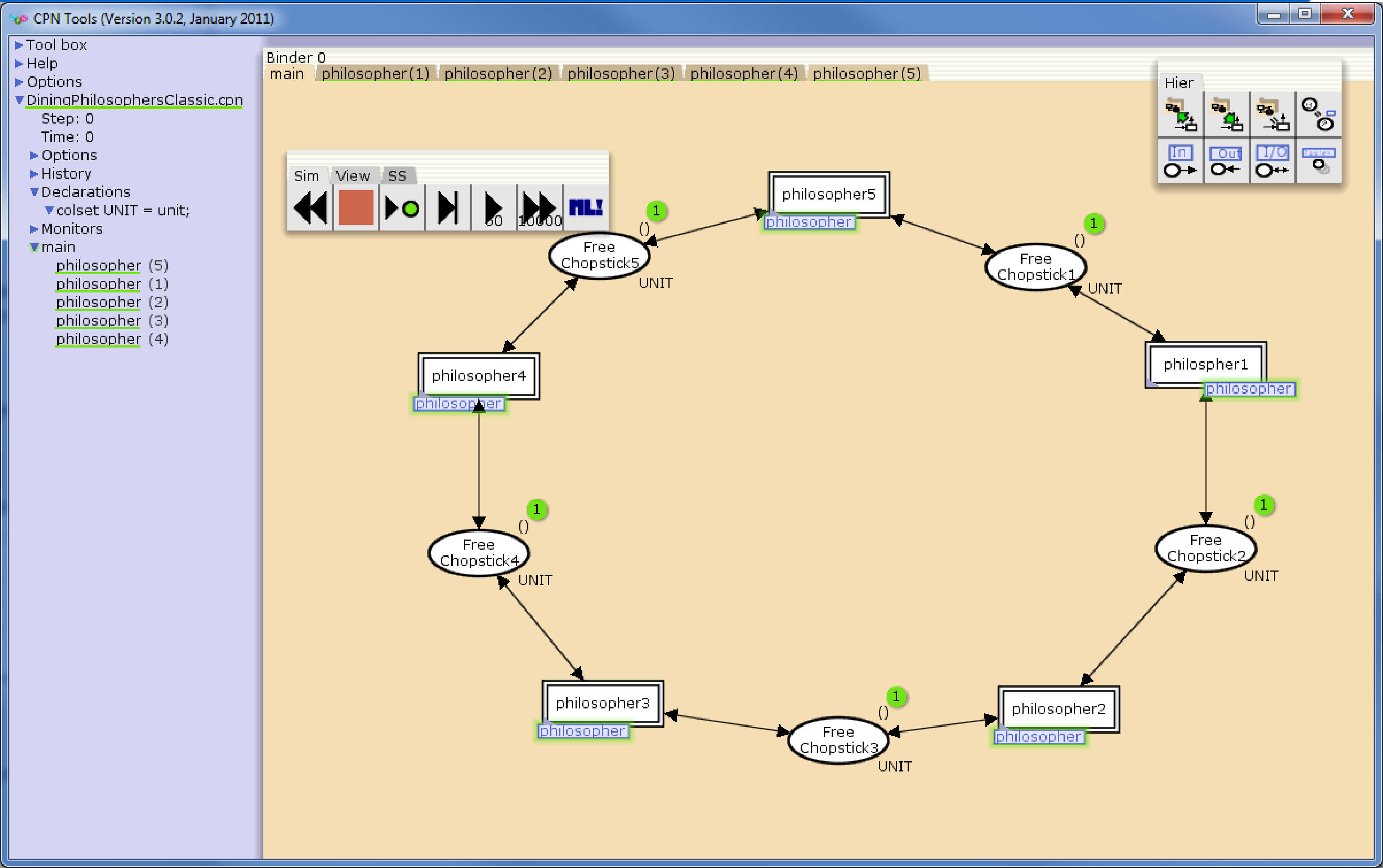
# Flat model is obtained by replacing substitution transitions by subpages

color BlackToken = unit;
var b:BackToken

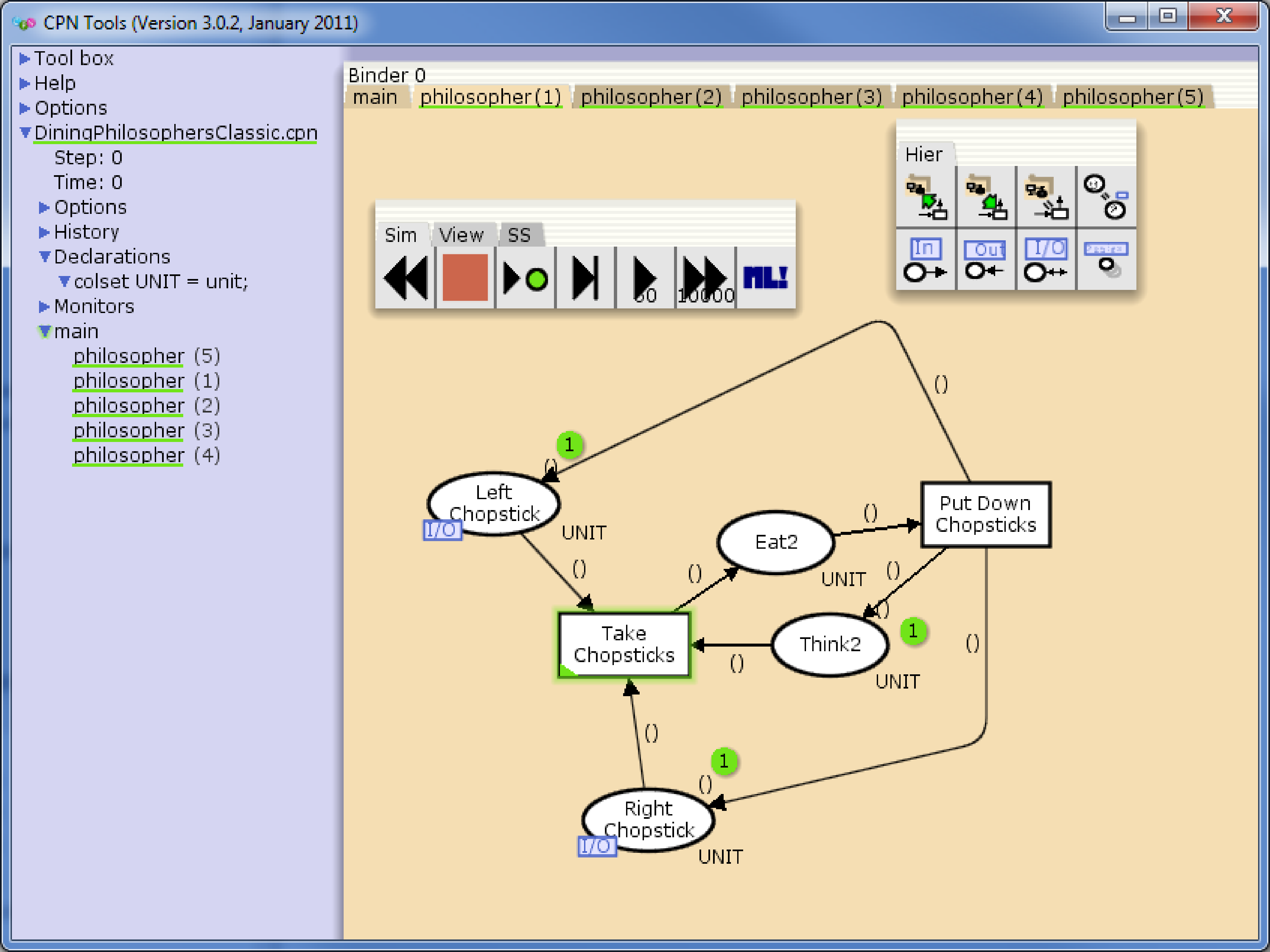


Repeat 5 times...

# In CPN Tools



CPN Tools (Version 3.0.2, January 2011)
Tool box
Help
Options
DiningPhilosophersClassic.cpn
Step: 0
Time: 0
Options
History
Declarations
colset UNIT = unit;
Monitors
main
philosopher (5)
philosopher (1)
philosopher (2)
philosopher (3)
philosopher (4)
Binder 0
main
philosopher(1)
philosopher(2)
philosopher(3)
philosopher(4)
philosopher(5)
Sim
View
SS
50
10000
ML!
Hier
In
Out
I/O
philosopher5
philosopher
philosopher4
philosopher
philosopher3
philosopher
philosopher2
philosopher
philospher1
philosopher
Free Chopstick5
UNIT
1
()
Free Chopstick1
UNIT
1
()
Free Chopstick2
UNIT
1
()
Free Chopstick3
UNIT
1
()
Free Chopstick4
UNIT
1
()



colset UNIT = unit;