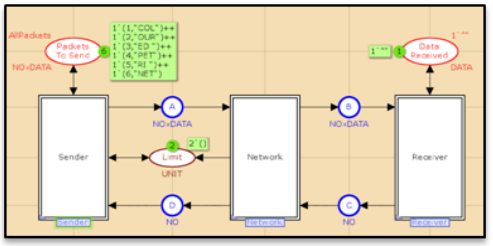
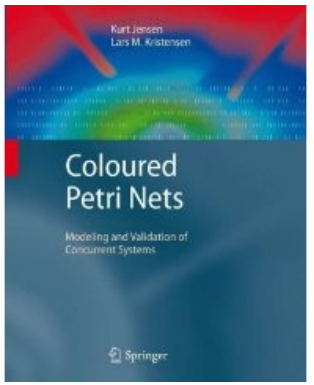
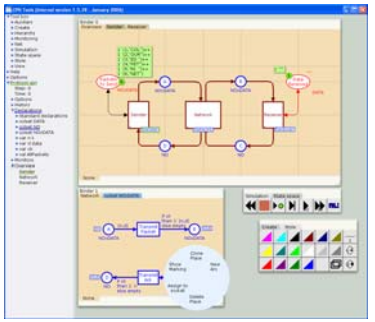
Coloured Petri Nets



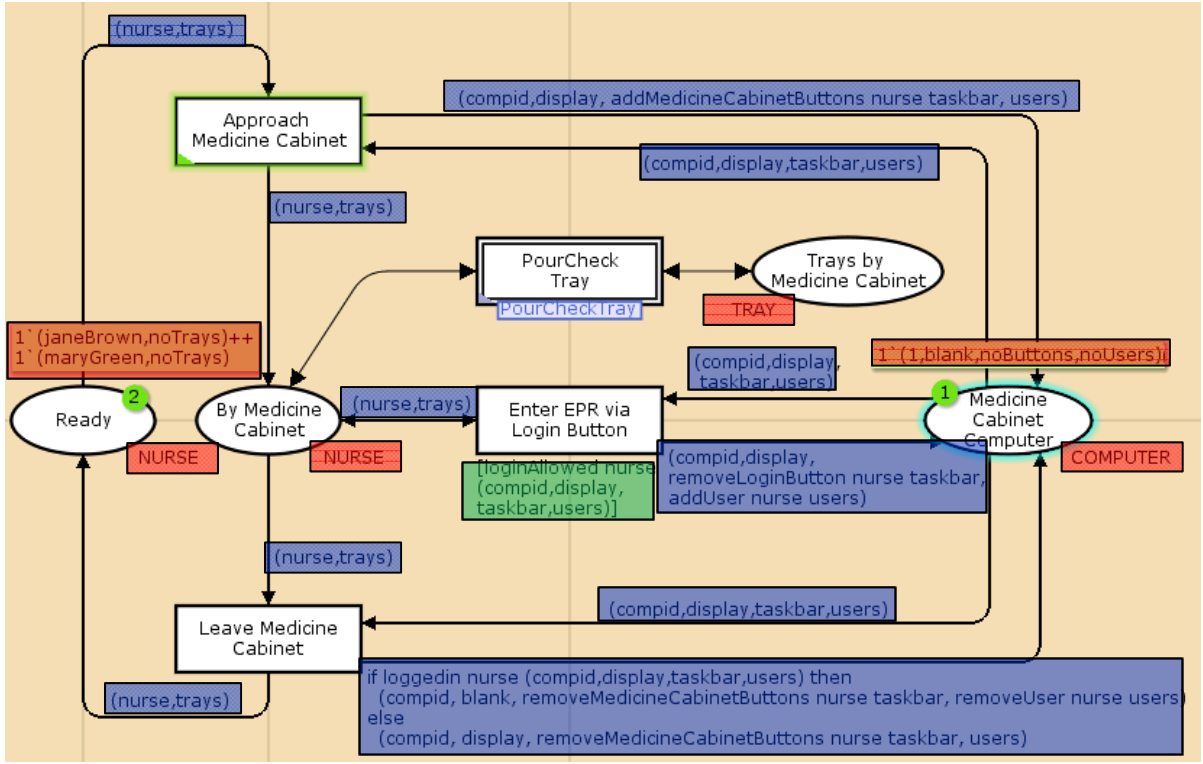




Lars M. Kristensen
Department of Computer Engineering
Bergen University College, NORWAY
Email: lmkr@hib.no / Web: www.hib.no/ansatte/lmkr

Coloured Petri Nets (CPNs)

- **Petri Nets and a programming language:**



Petri Nets:

- concurrency
- control structures
- synchronisation
- communication

Standard ML:

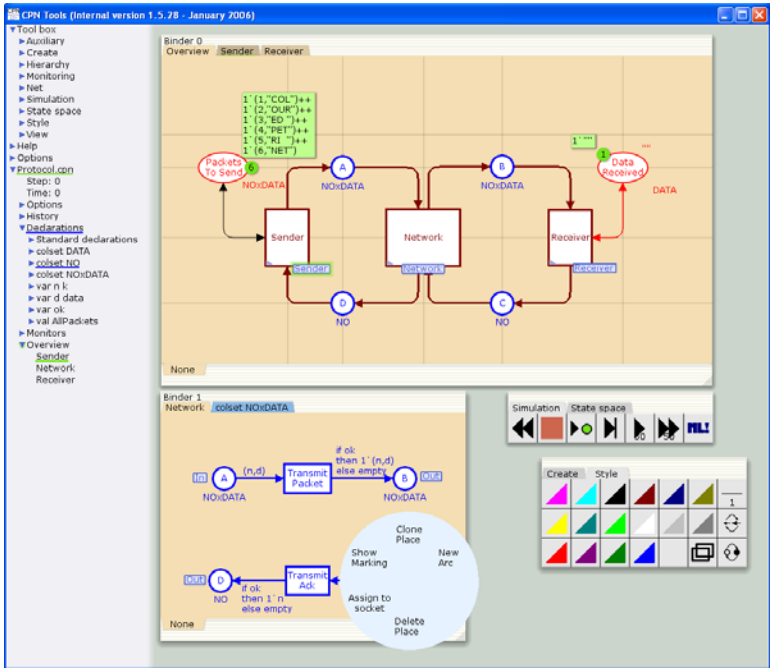
- Colour sets (data types) and markings
- Arc expressions
- Guards

- **Standard ML enables compact modelling and convenient modelling of data manipulation.**

CPN Tools

[www.daimi.au.dk/CPNTools]

- **Modelling and validation of Coloured Petri Net models are supported by CPN Tools:**
  - Editing and syntax check.
  - Interactive- and automatic simulation.
  - State space exploration and verification.
  - Performance analysis.
  - Behavioural visualisation using application domain graphics.

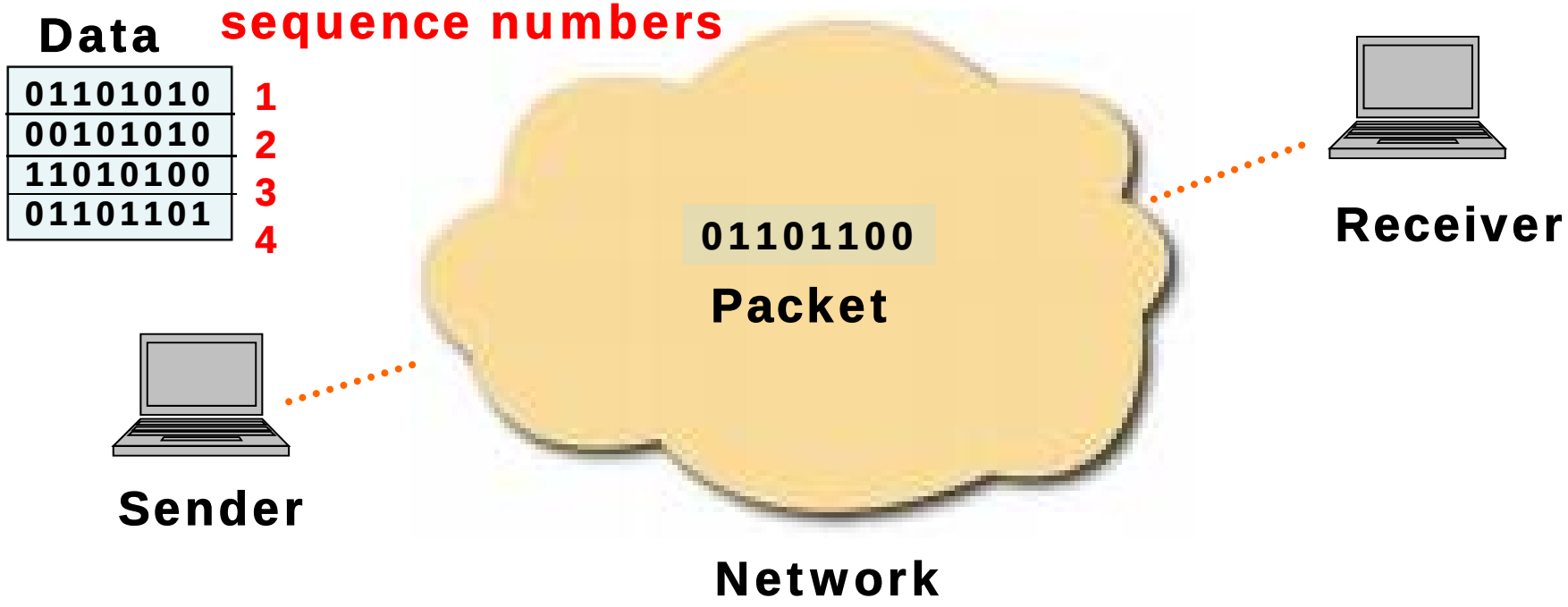


- **Currently 8000+ CPN Tools license holders in 130+ countries.**

Example: A Simple Communication Protocol

The Problem – in Greendale

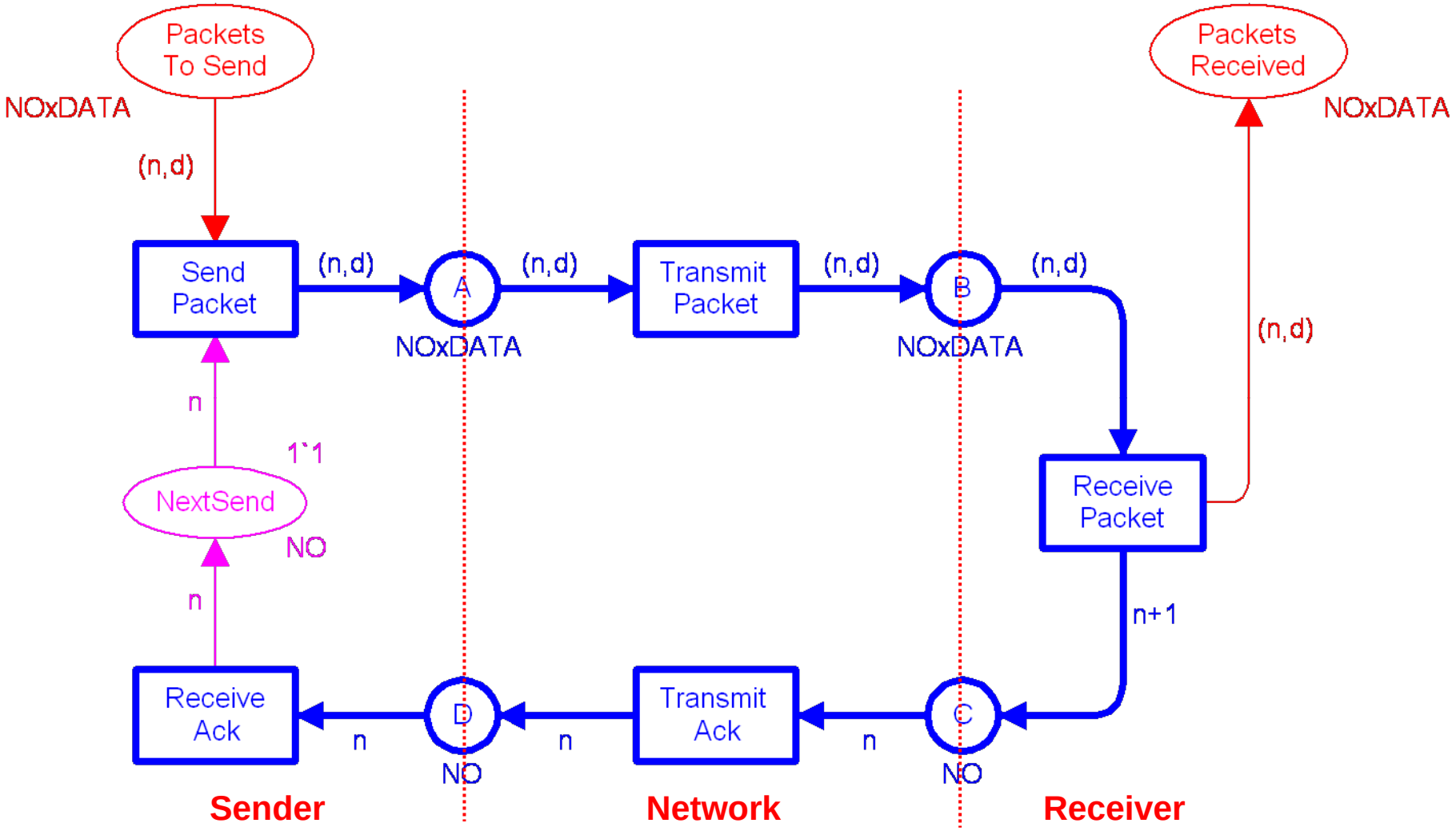
The Problem – in Data Networks

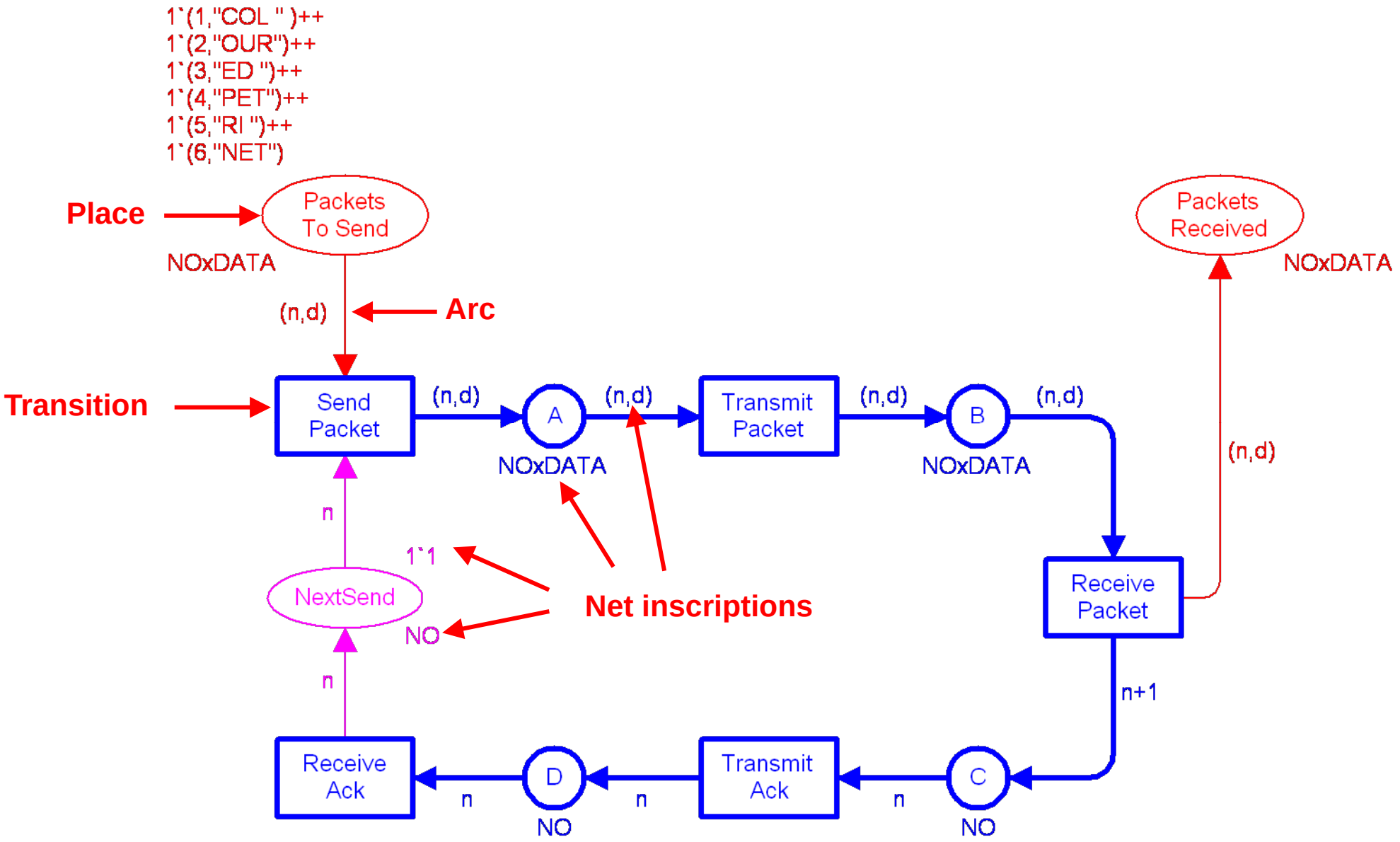


- Receiver must assemble original data.
- Stop-and-wait protocol: transmit one data packet at a time and wait for a matching acknowledgement.
- Initially we will assume a reliable network (no loss).

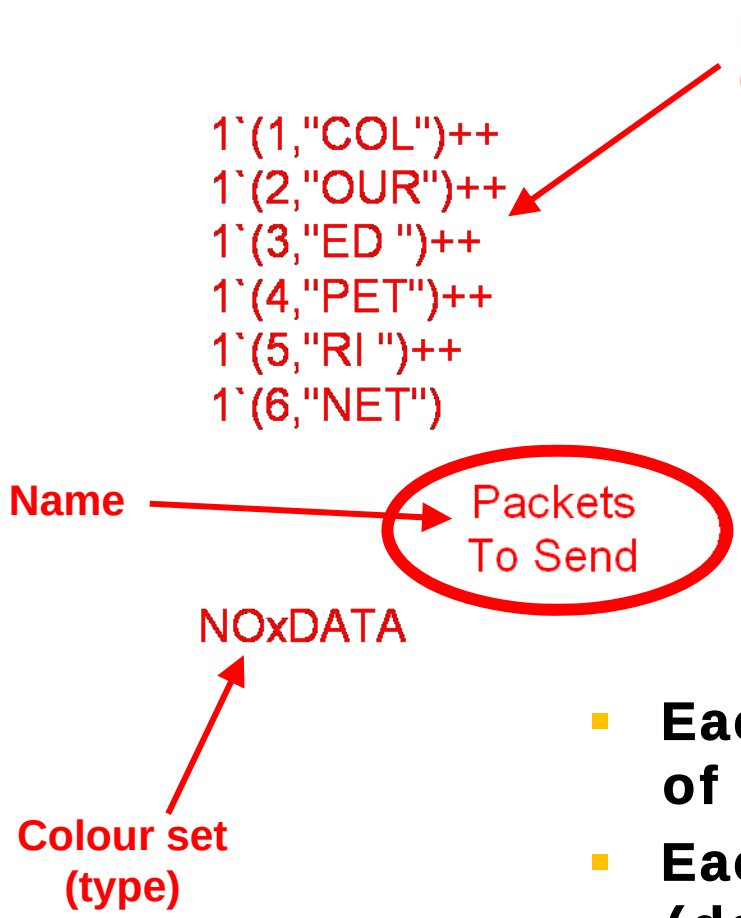
Part 1:
Basic Protocol CPN Model

The Coloured Petri Net Model




1`(1,"COL ")++
1`(2,"OUR")++
1`(3,"ED ")++
1`(4,"PET")++
1`(5,"RI ")++
1`(6,"NET")
Place
Packets To Send
NOxDATA
(n,d)
Arc
Transition
Send Packet
(n,d)
A
NOxDATA
(n,d)
Transmit Packet
(n,d)
B
NOxDATA
(n,d)
Packets Received
NOxDATA
(n,d)
Receive Packet
n+1
Transmit Ack
C
NO
n
Receive Ack
D
NO
n
n
n
NextSend
1`1
NO
n
Net inscriptions

Places model the state of the system

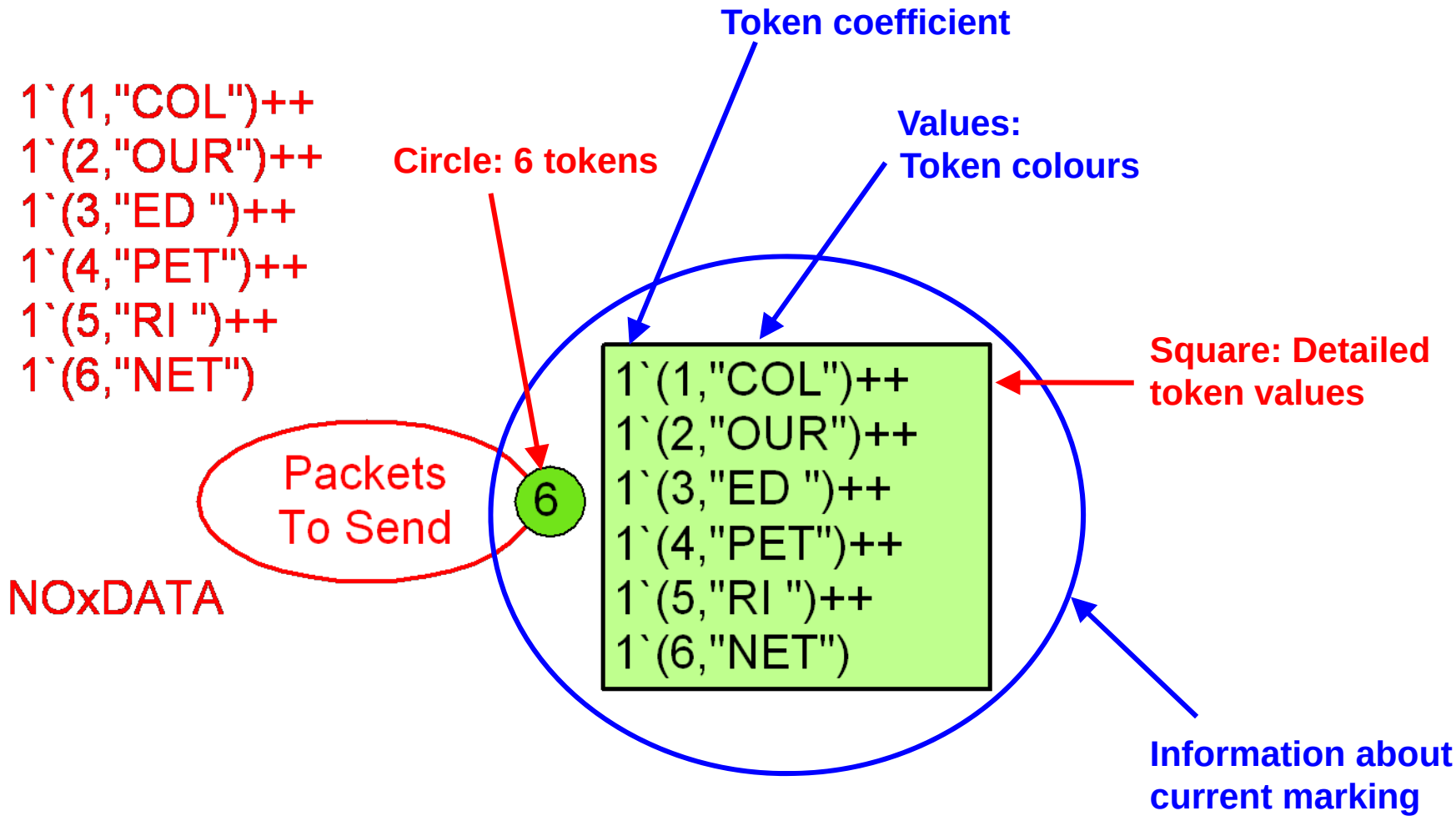


Definition of colour sets:

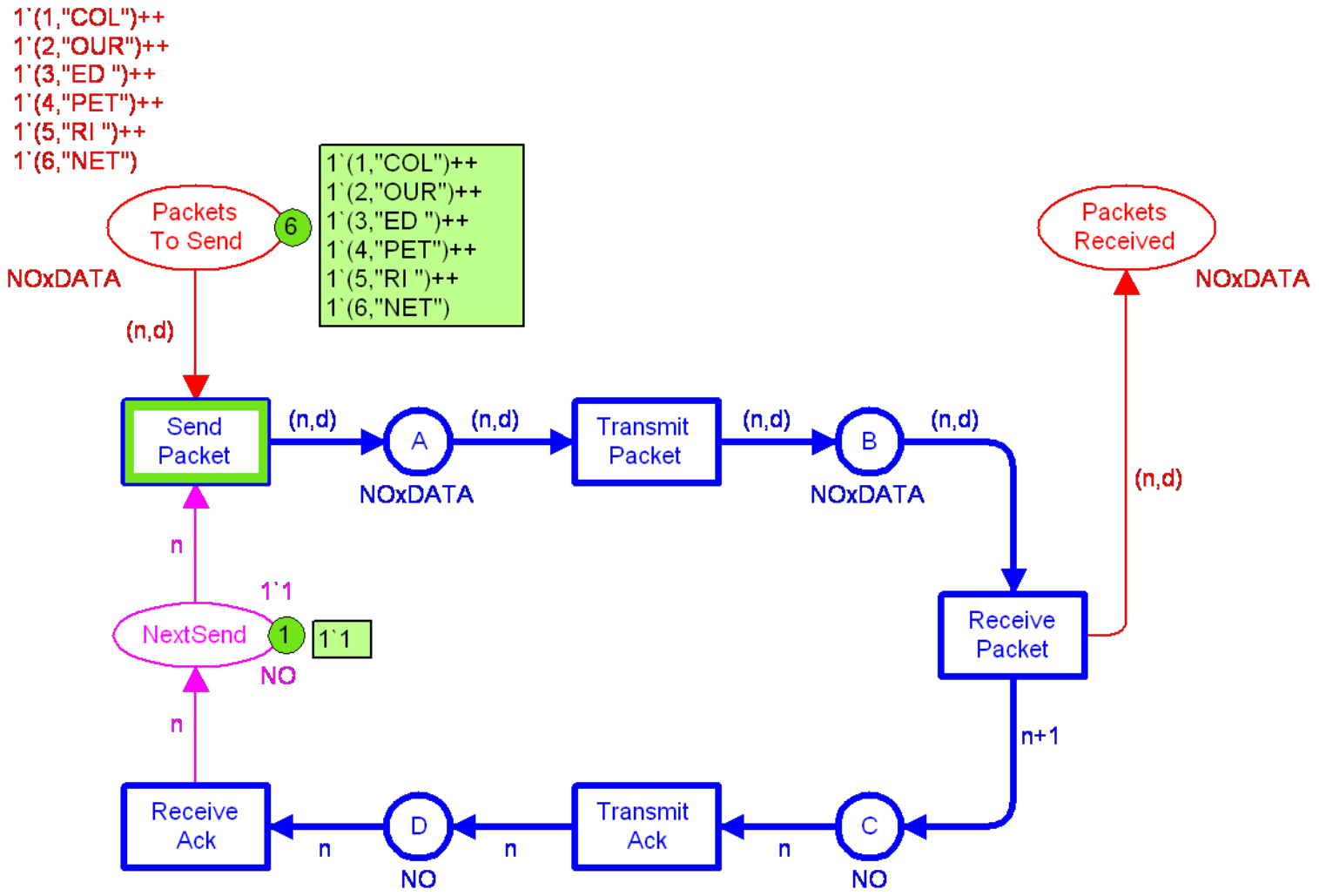
```
colset NO        = int;       (* integers *)
colset DATA      = string;    (* text strings *)
colset NOxDATA = product NO * DATA;
```

- **Each place may contain a number of tokens.**
- **Each token carries a colour (data value).**
- **The colour set of a place specifies the set of allowed token colours.**

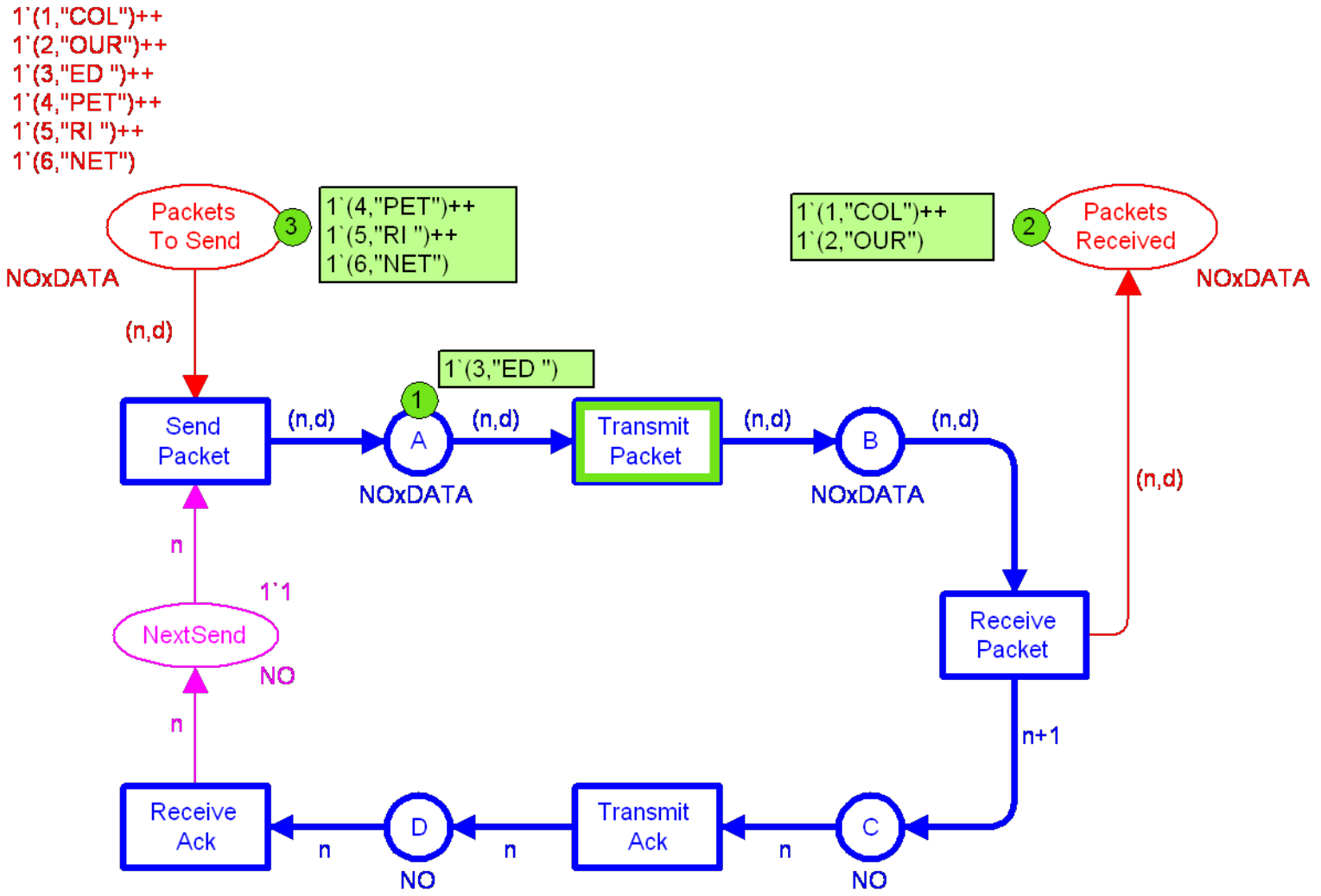
Current marking



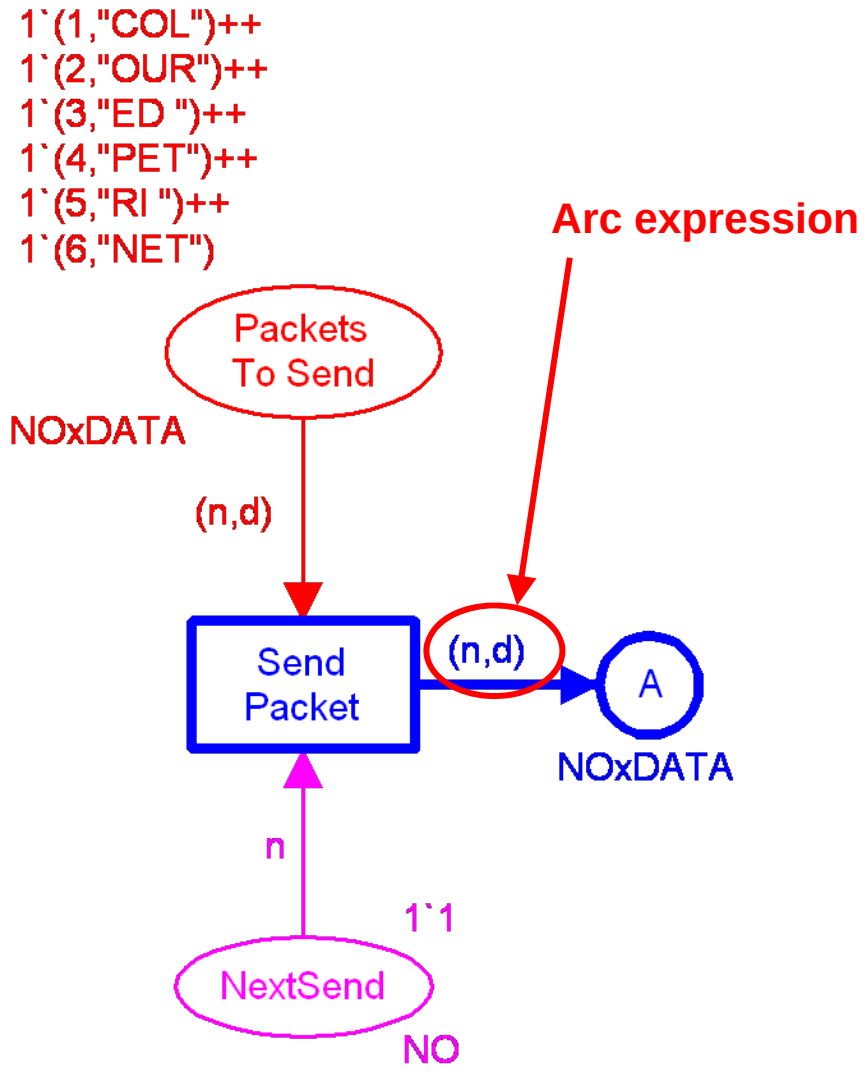
Initial marking of CPN model



Intermediate marking of model



Transitions model events of the system



The type of the arc expression must match the colour set of the attached place (or the multi-set type over the colour set)

Declaration of variables:

```
var n : NO;      (* integers *)
var d : DATA;    (* strings *)
```

Binding of transition variables:

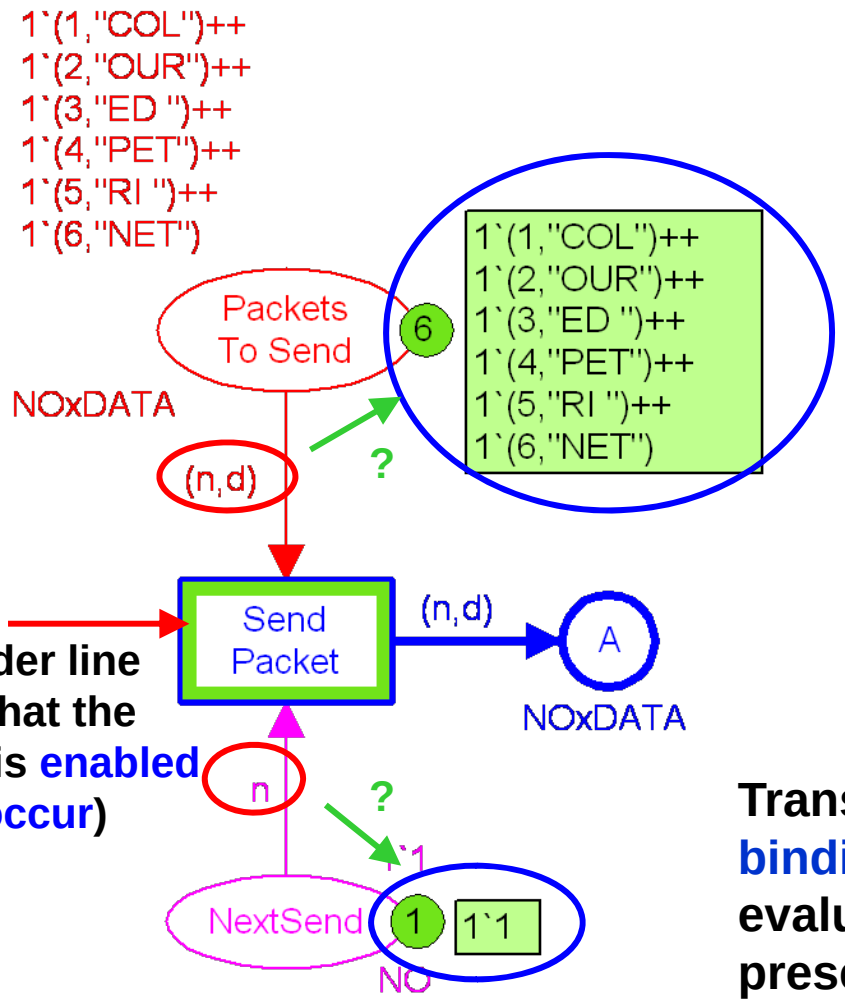
<n=3,d="CPN">

Evaluation of expressions:

(n,d) → (3,"CPN") : NOxDATA

n → 3 : NO

Enabling of transitions



Two variables:

```
var n : NO;      (* integers *)
var d : DATA;    (* strings *)
```

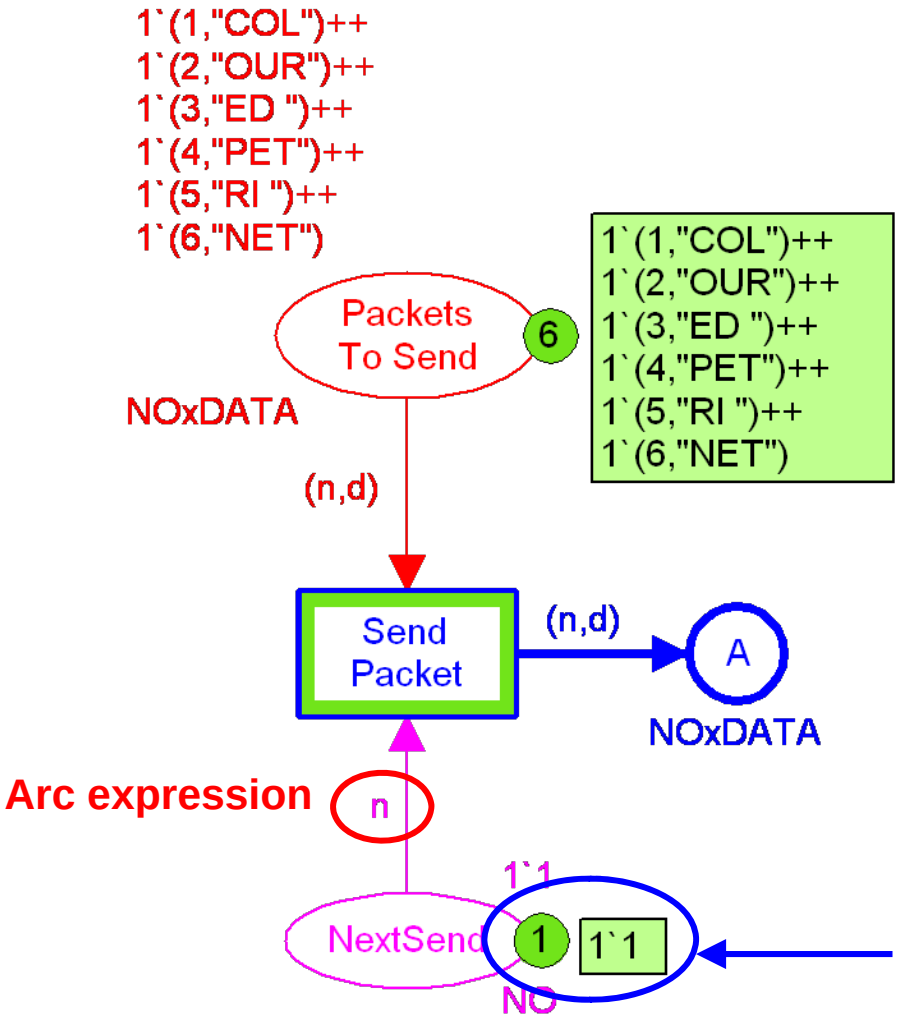
Binding: < n=? , d=? >

NO DATA

Green border line indicates that the transition is enabled (ready to occur)

Transition is enabled if we can find a binding so that each input arc expression evaluates to a multi-set of tokens that are present on the corresponding input place.

Enabling of SendPacket

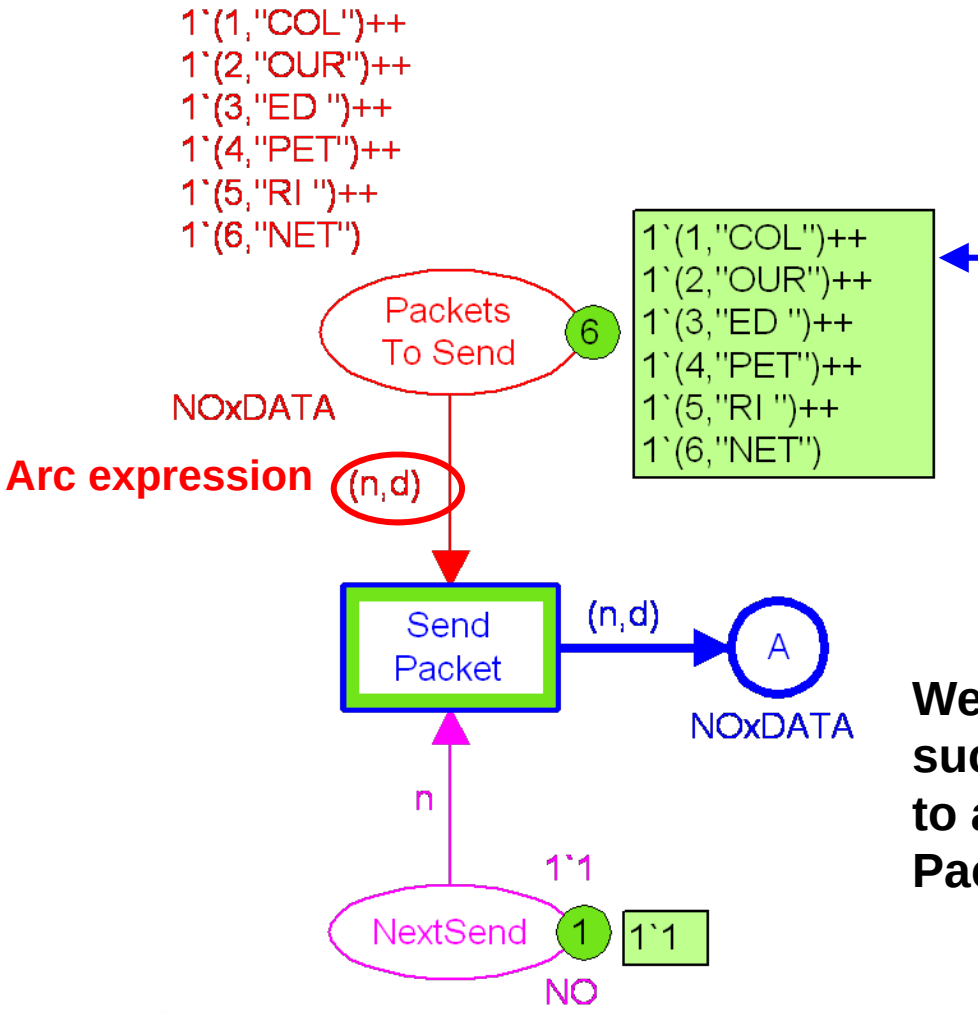


Binding: < n=1 , d=? >

We want to find a binding for the variable n such that the arc expression n evaluates to a multi-set of colours which is present on the place NextSend.

One token with value 1

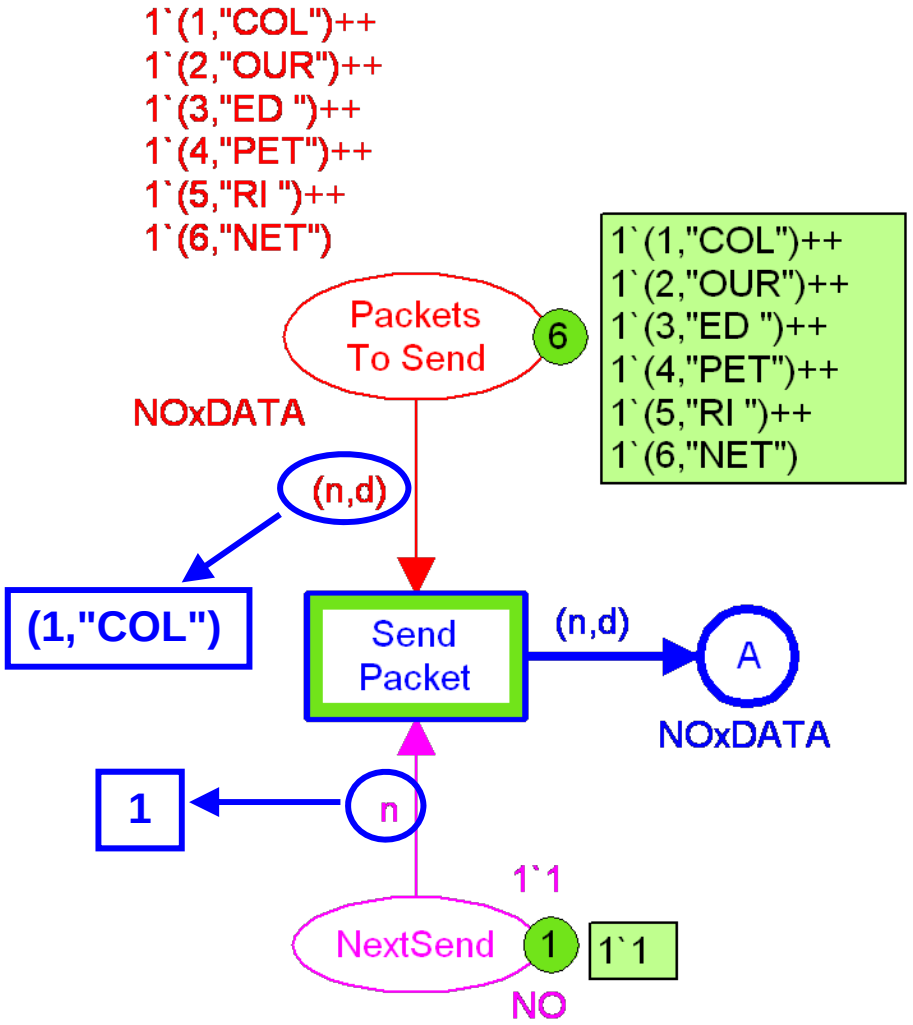
Enabling of SendPacket



Binding: < n=1 , d="COL" >

We want to find a binding for the variable d such that the arc expression (n,d) evaluates to a multi-set which is present on the place PacketsToSend.

Enabling of SendPacket

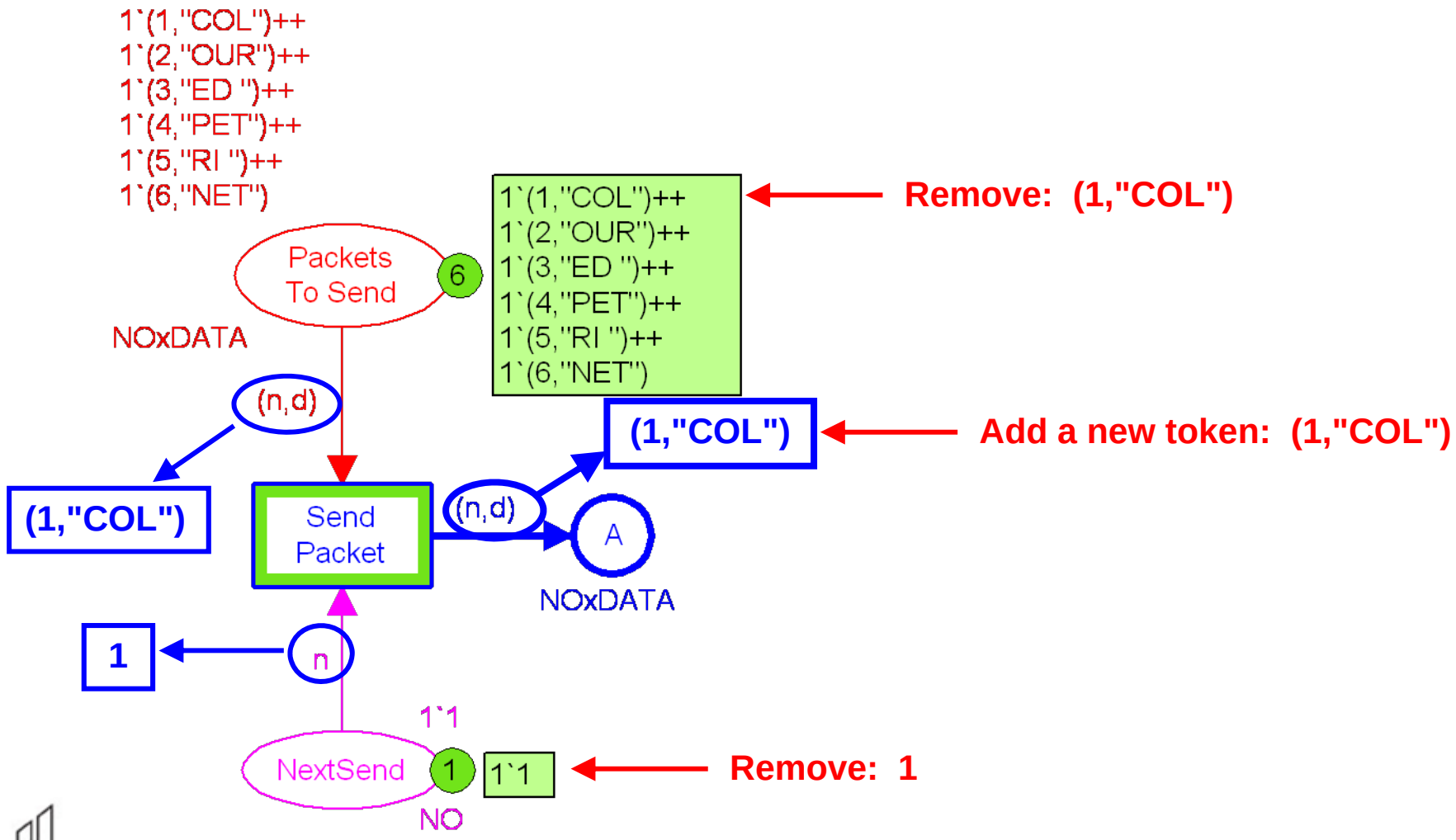


We have found a binding so that each input arc expression evaluates to a colour that is present on the corresponding input place

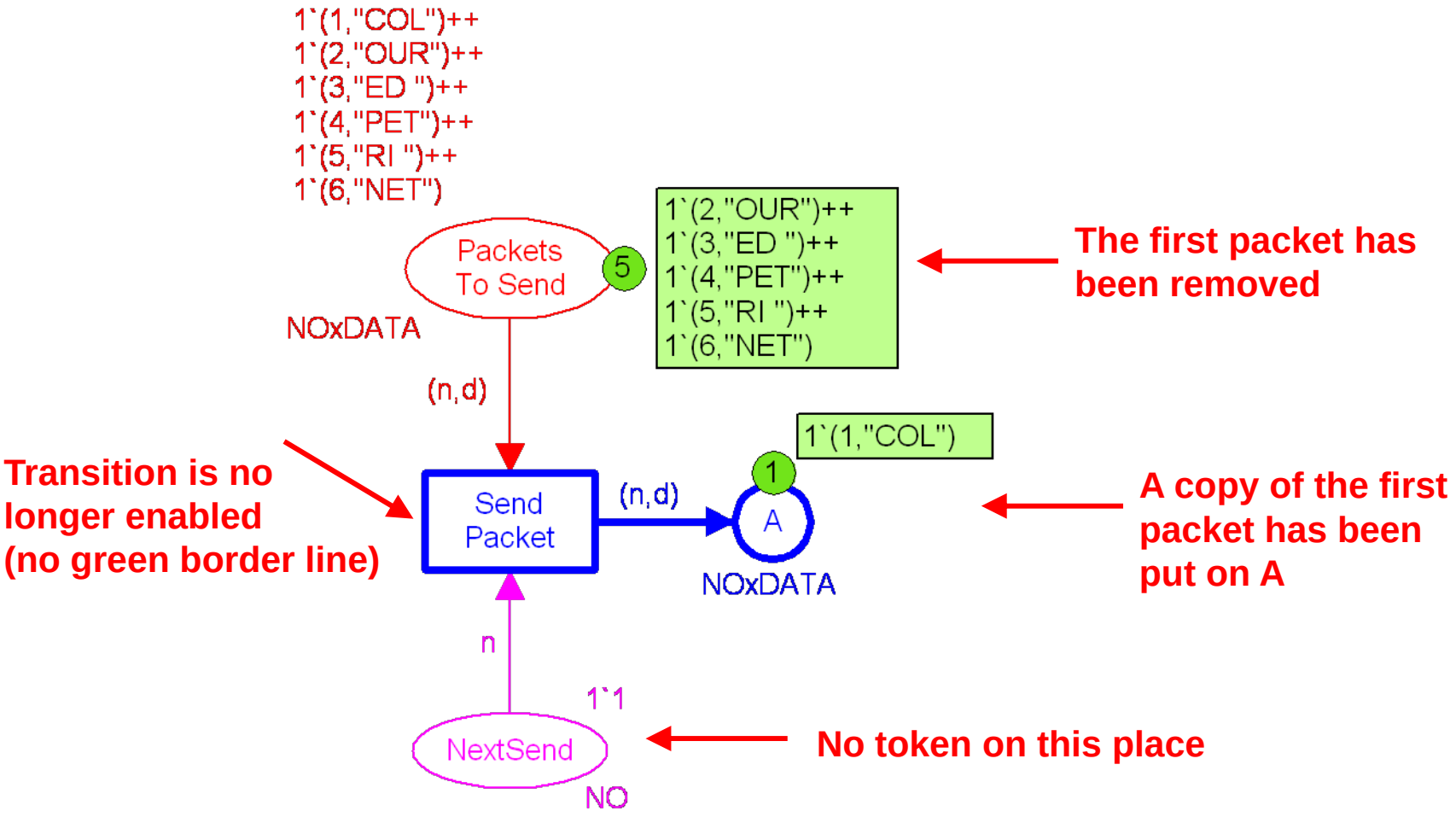
Binding: < n=1 , d="COL" >

Transition is enabled (ready to occur)

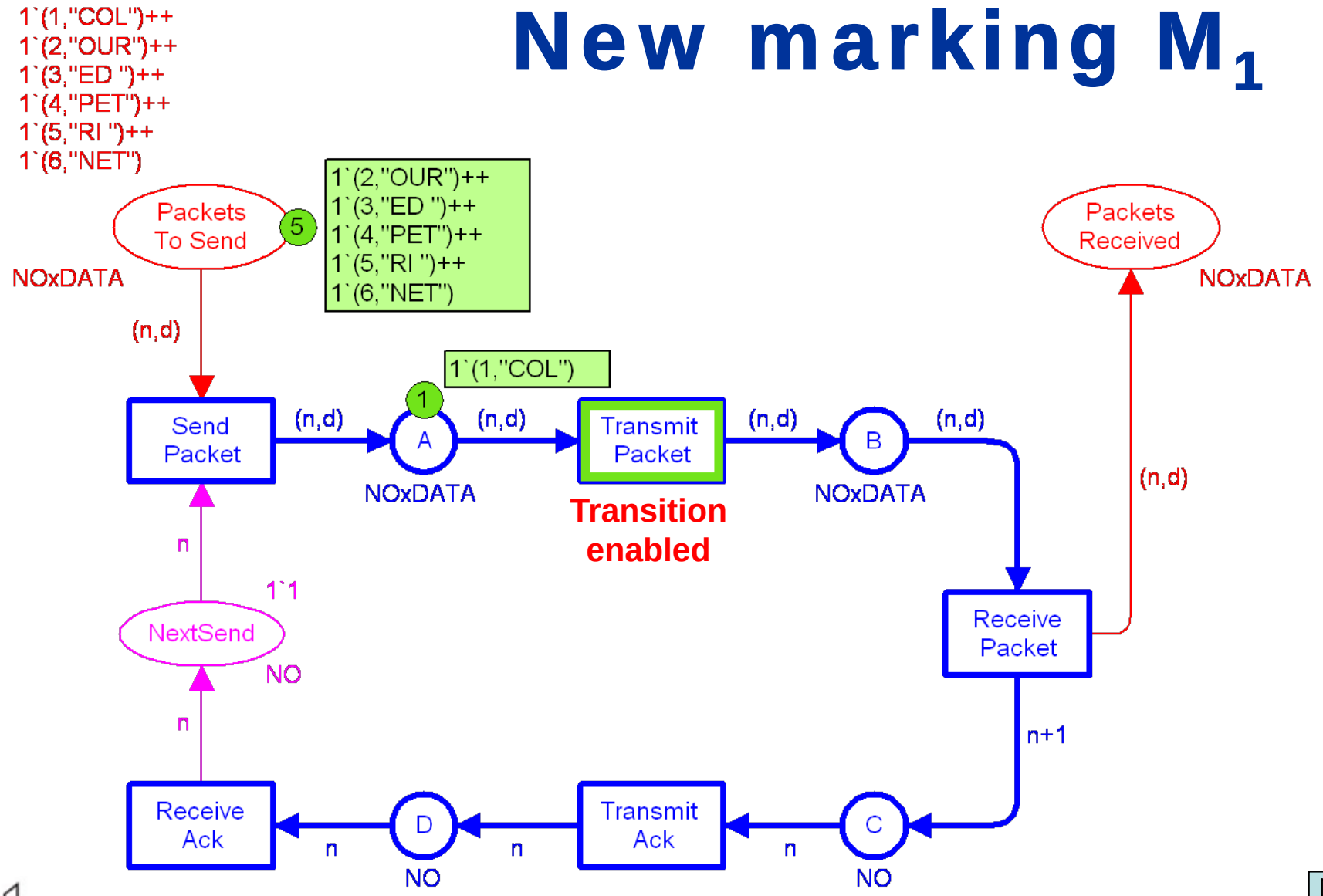
Occurrence of SendPacket in binding <n=1,d="COL">



New marking after occurrence of SendPacket in binding <n=1,d="COL">

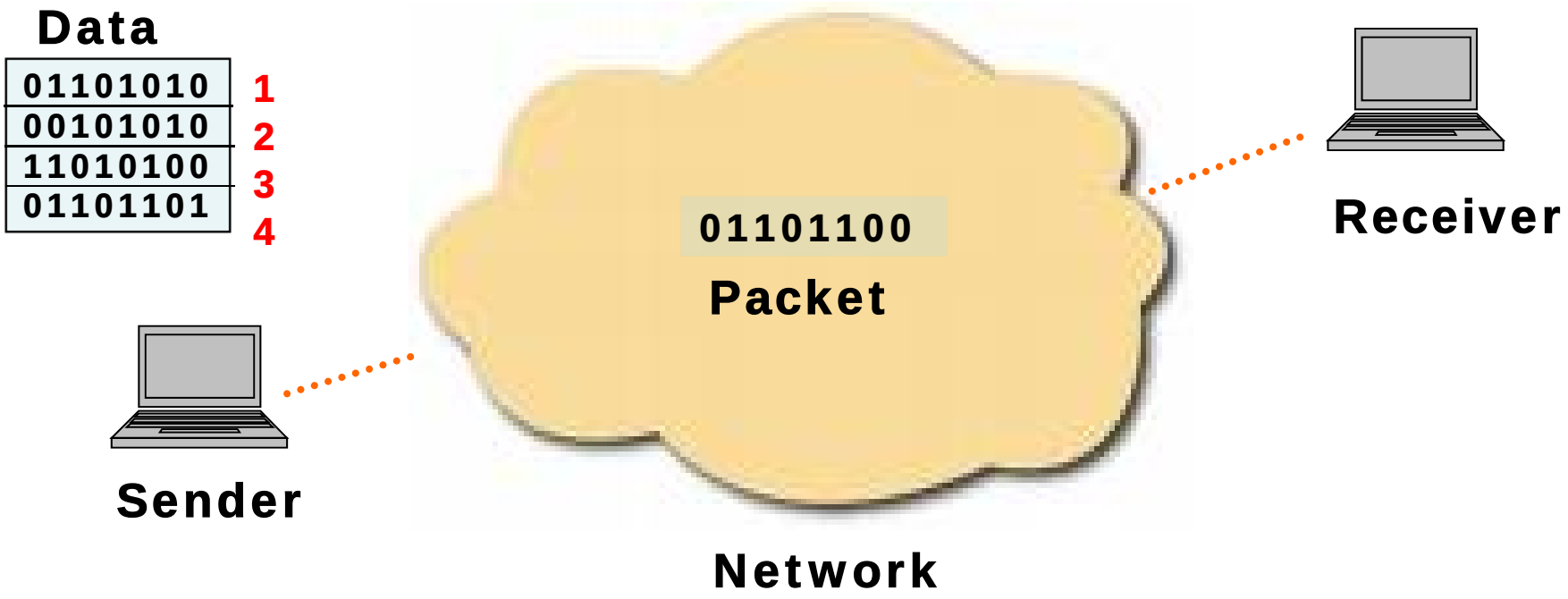


New marking M_1



Part 2:
Extended Protocol CPN Model

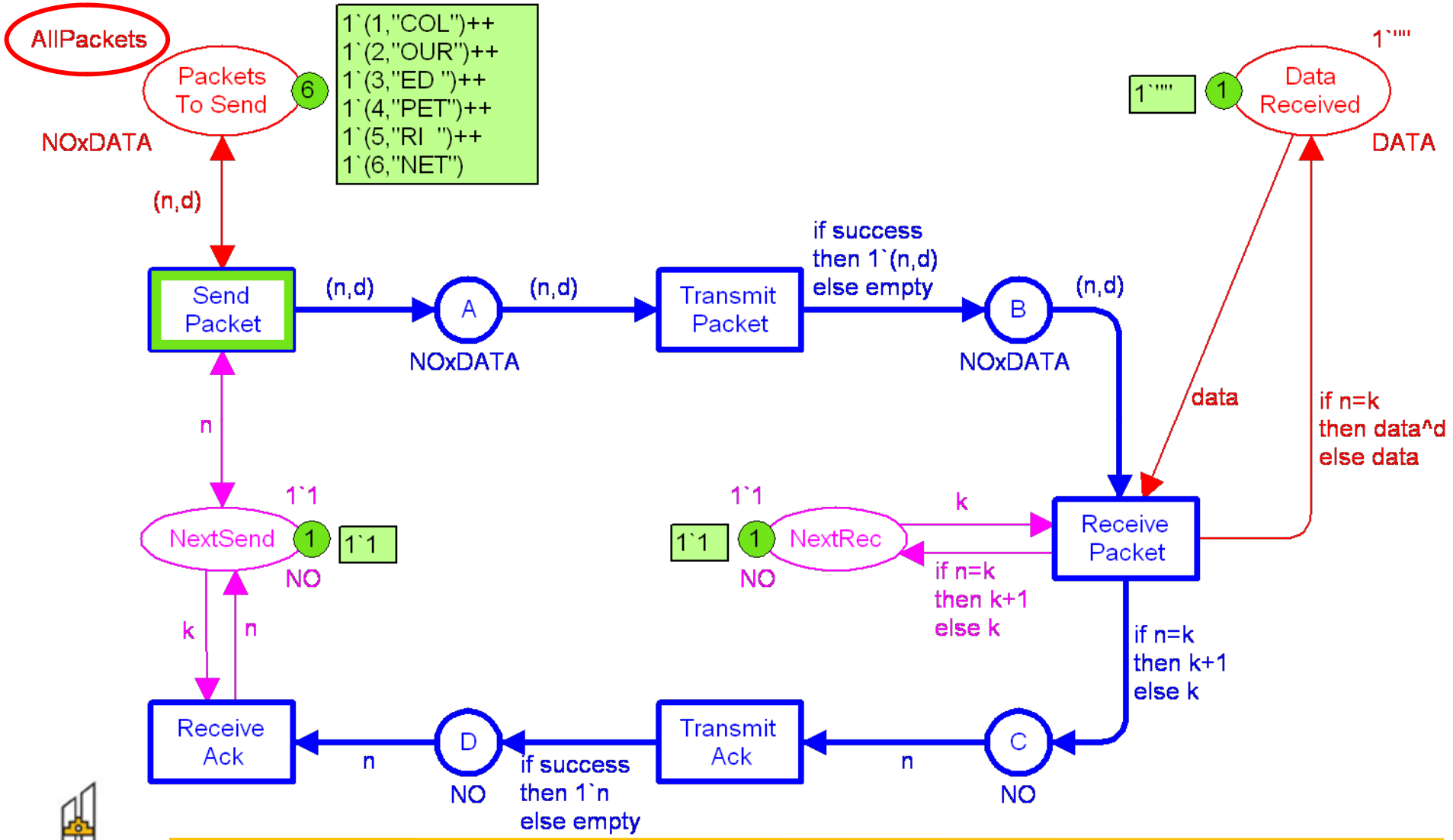
The Simple Protocol Revisited



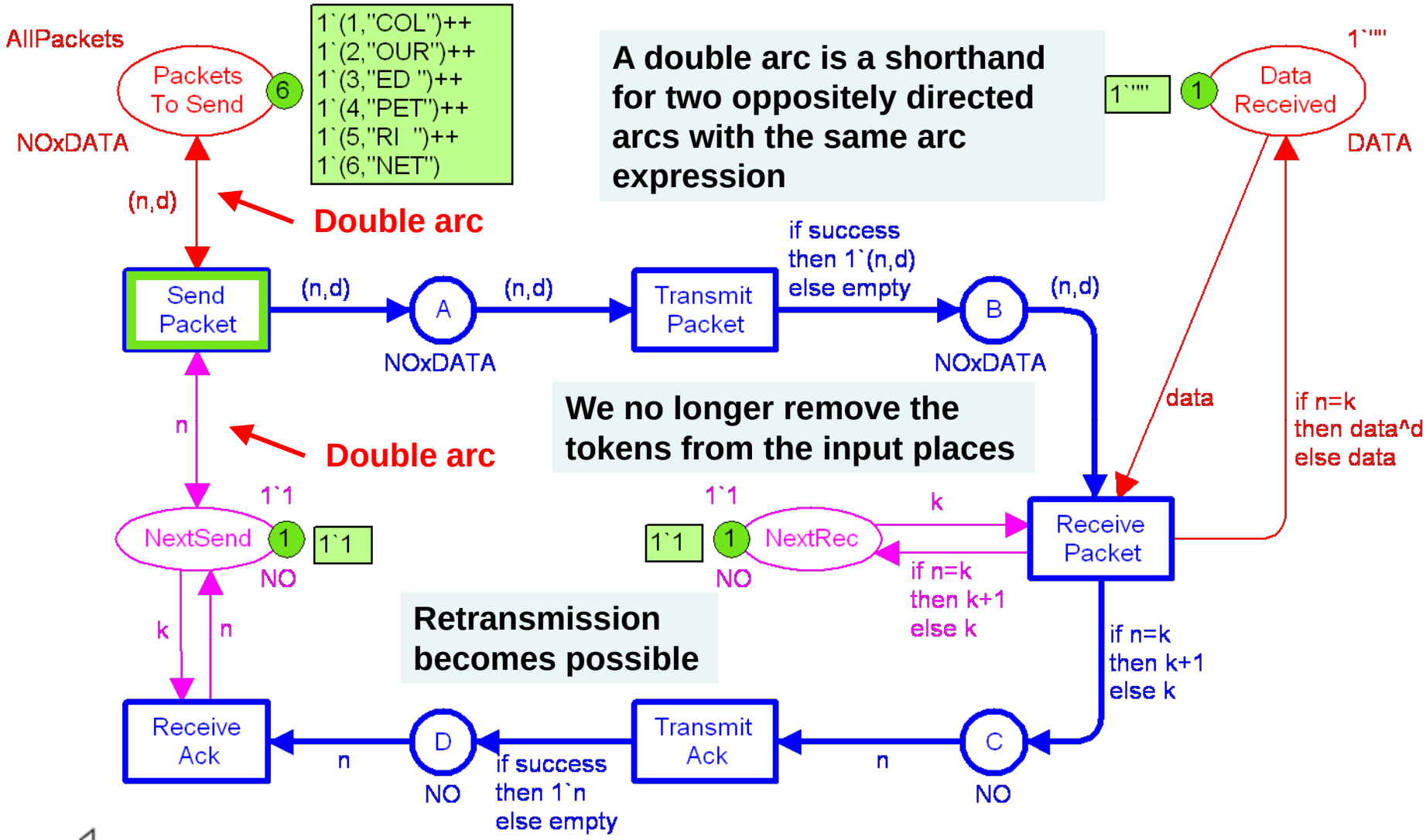
- **Unreliable network** (loss and overtaking).
- Sender must **retransmit packets** and keep track of the data packet currently being sent.
- Receiver keeps track of the **data packet expected next**.

Second Version

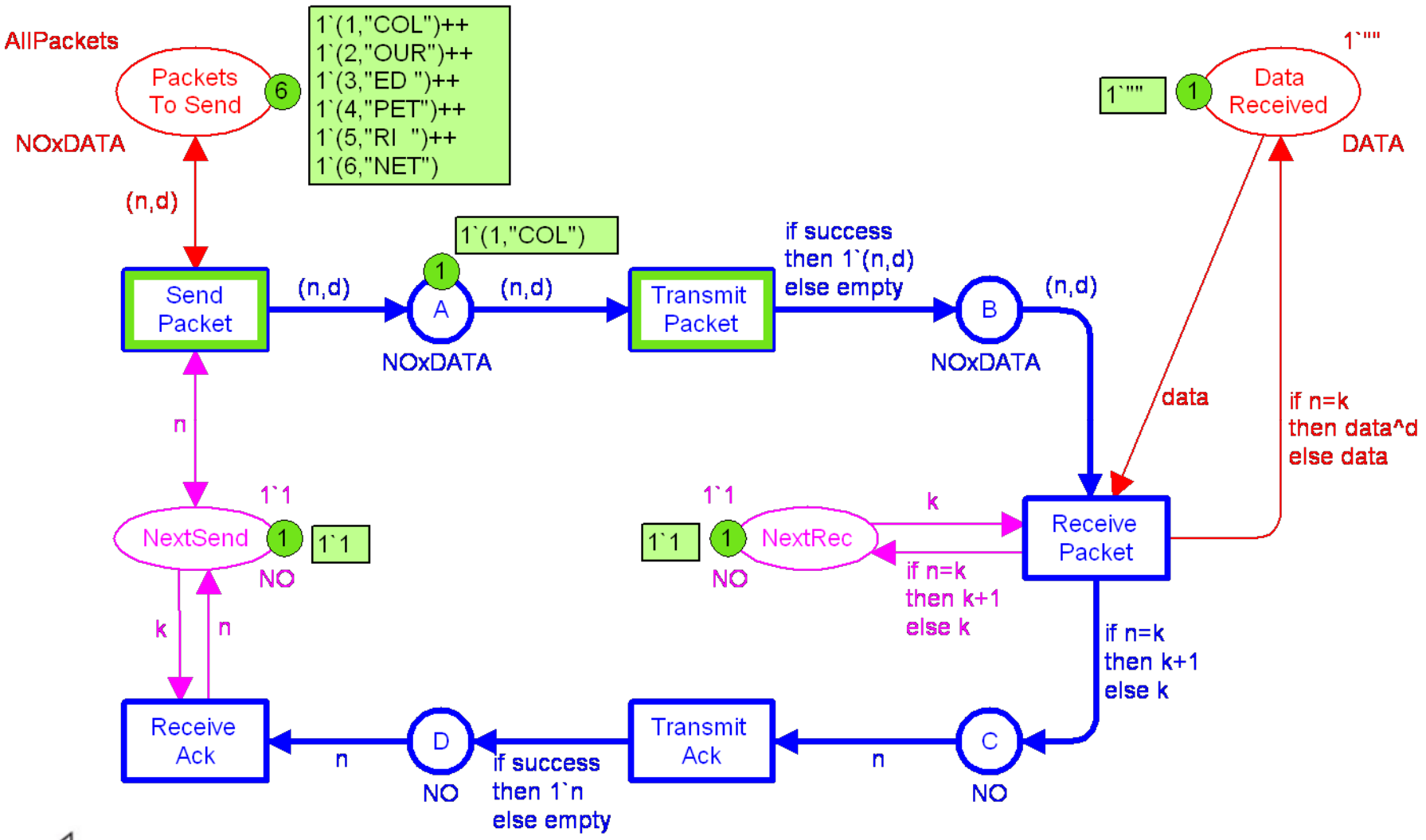
```
val AllPackets = 1‘(1,"COL") ++ 1‘(2,"OUR") ++
                 1‘(3,"ED ") ++ 1‘(4,"PET") ++
                 1‘(5,"RI ") ++ 1‘(6,"NET");
```



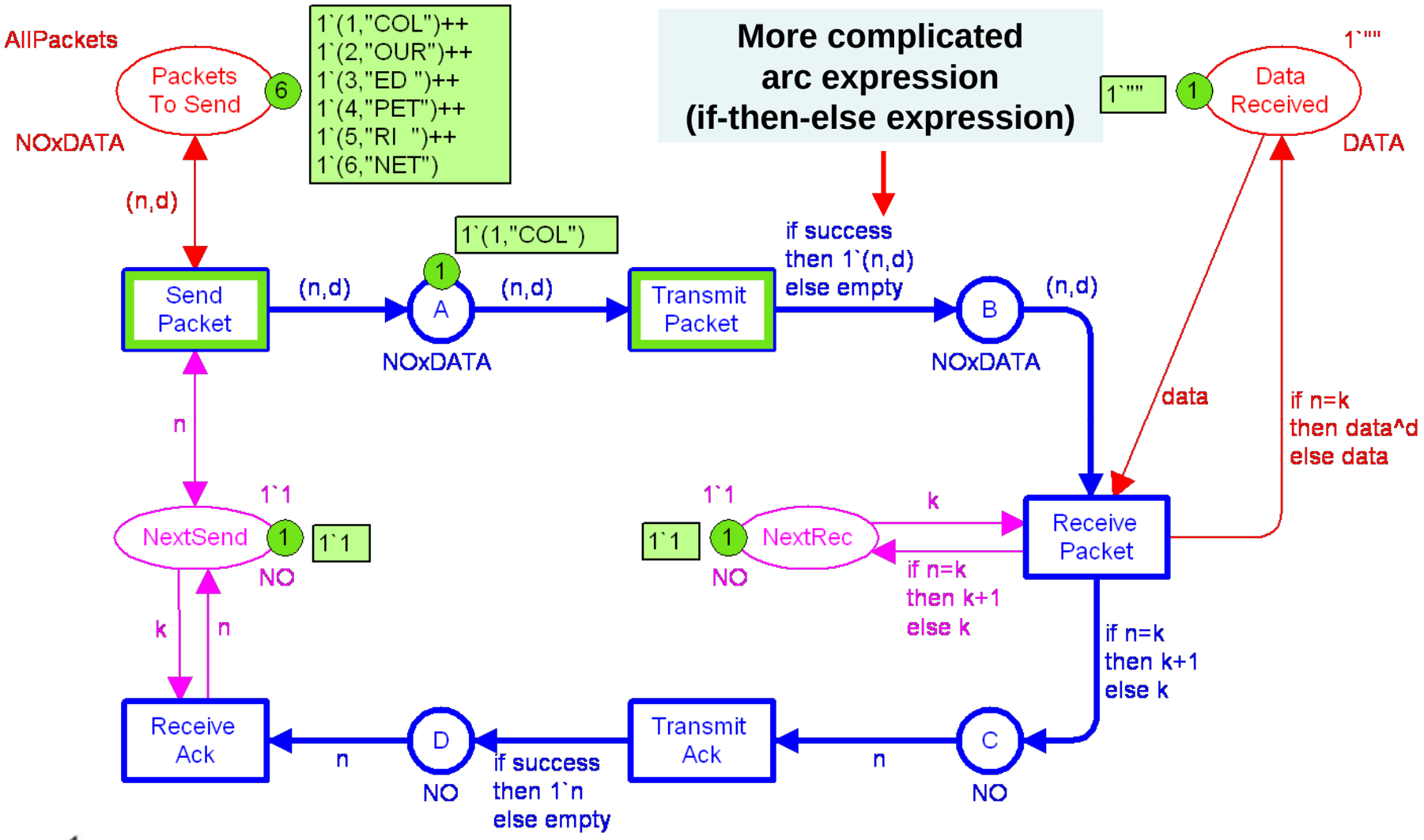
Double Arcs


AllPackets
Packets To Send
6
NOxDATA
1`(1,"COL")++
1`(2,"OUR")++
1`(3,"ED ")++
1`(4,"PET")++
1`(5,"RI ")++
1`(6,"NET")
(n,d)
Double arc
A double arc is a shorthand for two oppositely directed arcs with the same arc expression
1`""
1
Data Received
1`""
DATA
Send Packet
(n,d)
A
NOxDATA
(n,d)
Transmit Packet
if success then 1`(n,d) else empty
B
NOxDATA
(n,d)
data
if n=k then data^d else data
n
Double arc
We no longer remove the tokens from the input places
1`1
NextSend
1
1`1
NO
1`1
1`1
1
NextRec
NO
k
if n=k then k+1 else k
Receive Packet
k
n
Retransmission becomes possible
if n=k then k+1 else k
Receive Ack
n
D
NO
if success then 1`n else empty
Transmit Ack
n
C
NO

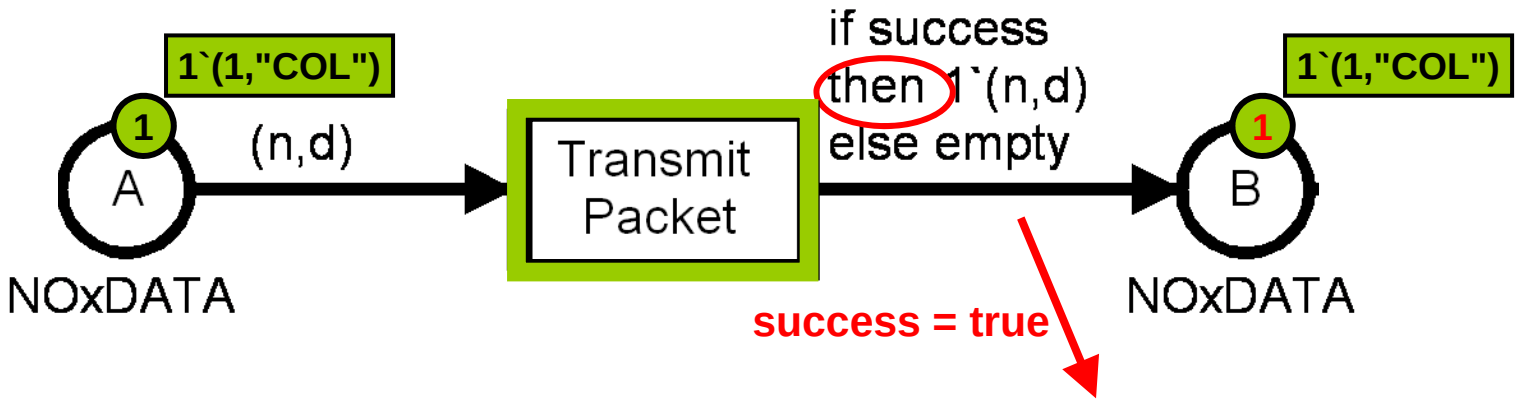
Occurrence of SendPacket <n=1,d="COL">



More Complex Arc Expression



If-then-else Expression



New variable:

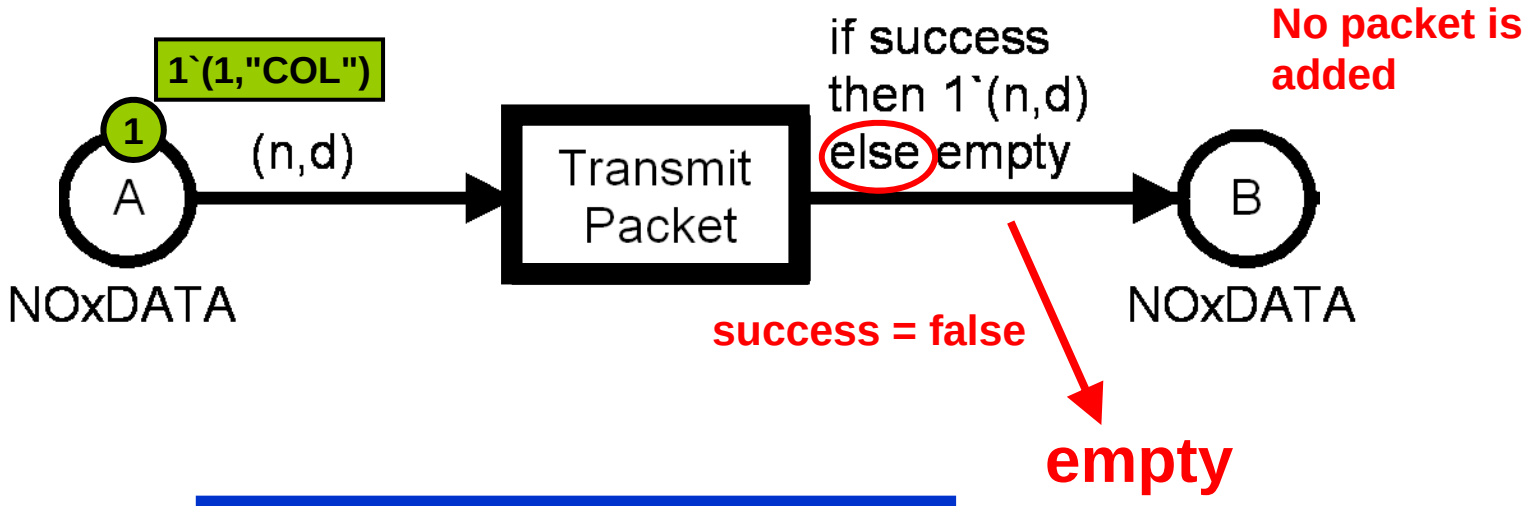
```
var success : BOOL;
```

1`(1,"COL")

Successful transmission over the network

b_{succ} = <n=1, d="COL", success=true>
b_{fail} = <n=1, d="COL", success=false>

If-then-else Expression

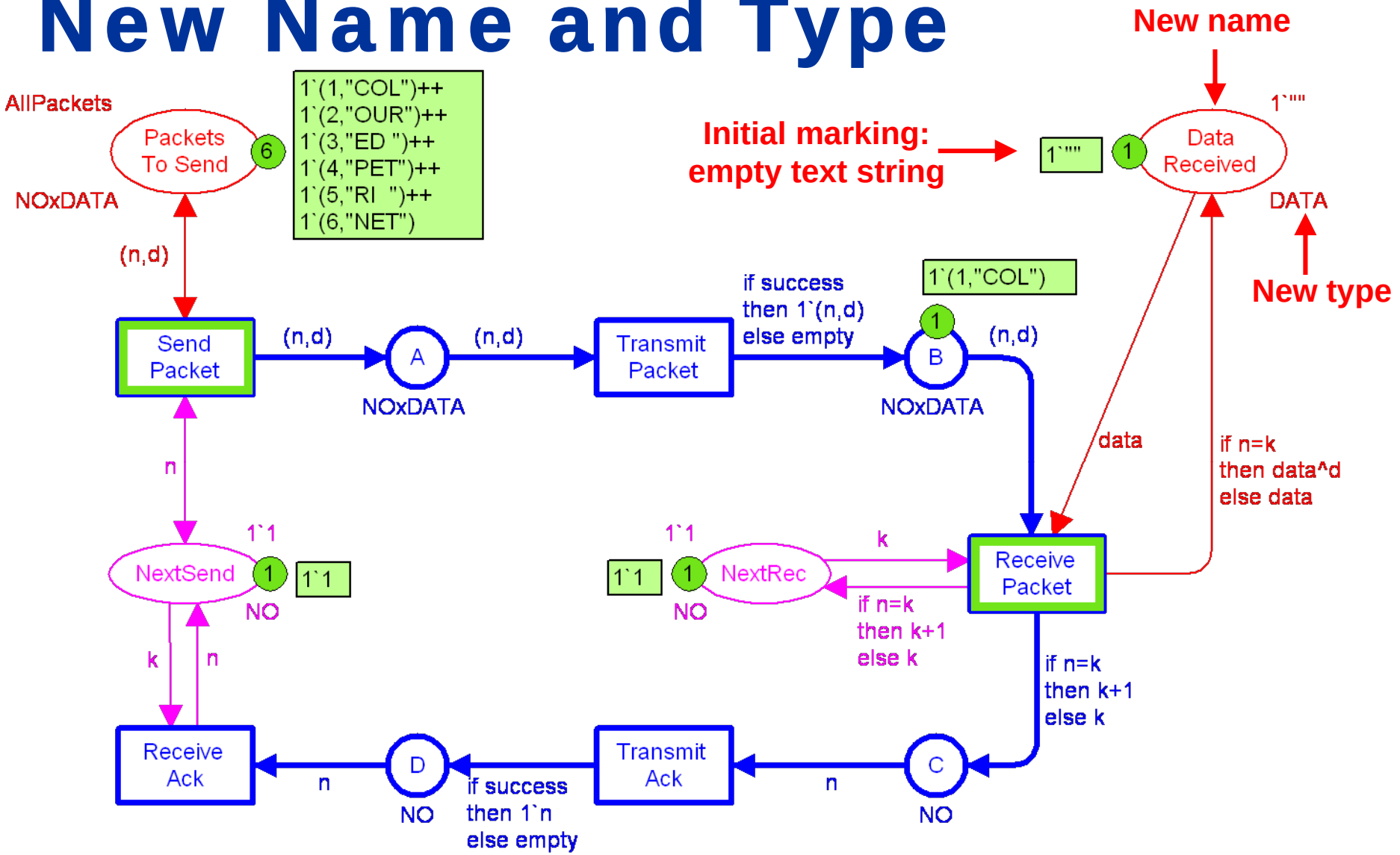



```
var success : BOOL;
```

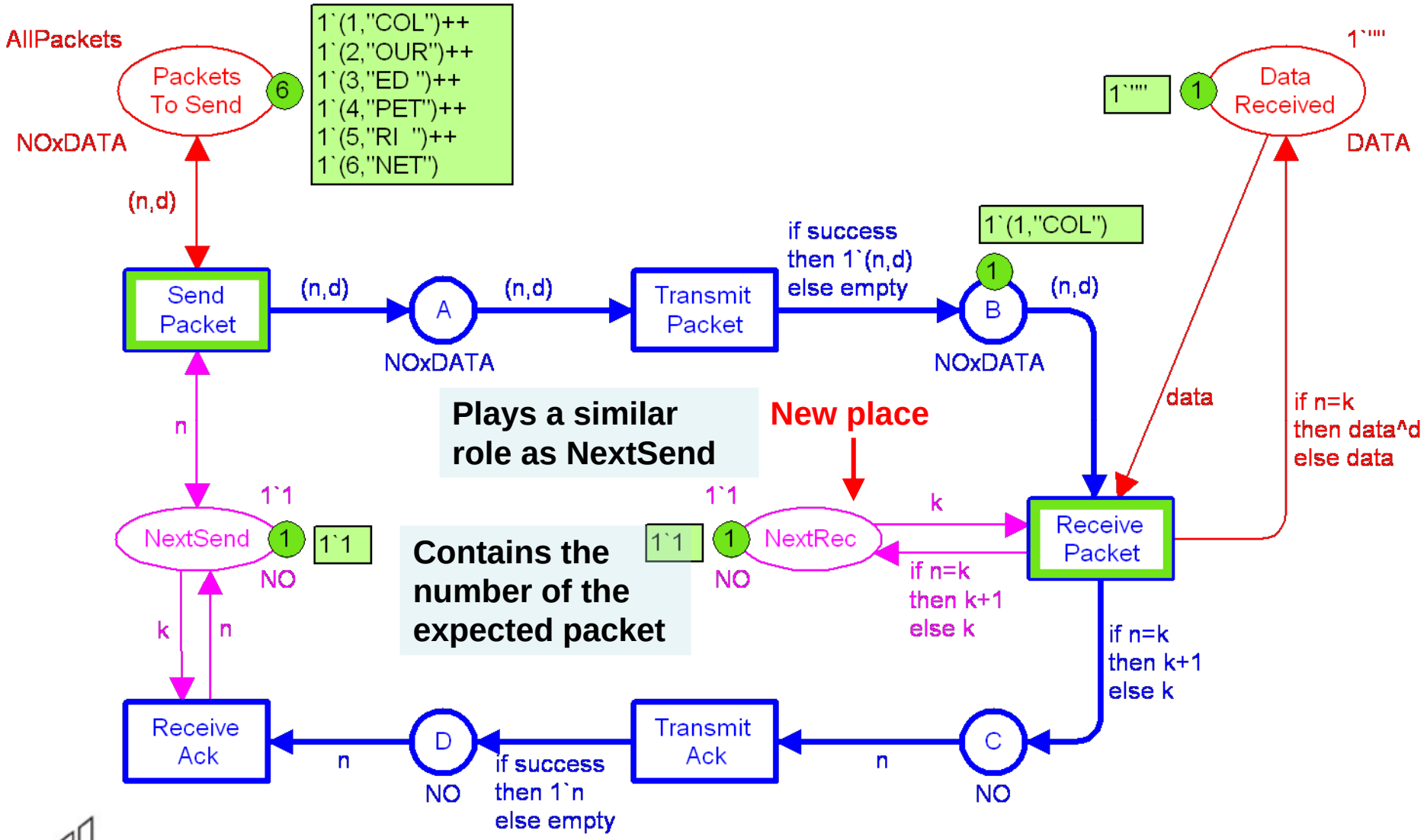
Packet is lost during transmission

b_{succ} = <n=1, d="COL", success=true>
b_{fail} = <n=1, d="COL", success=false>

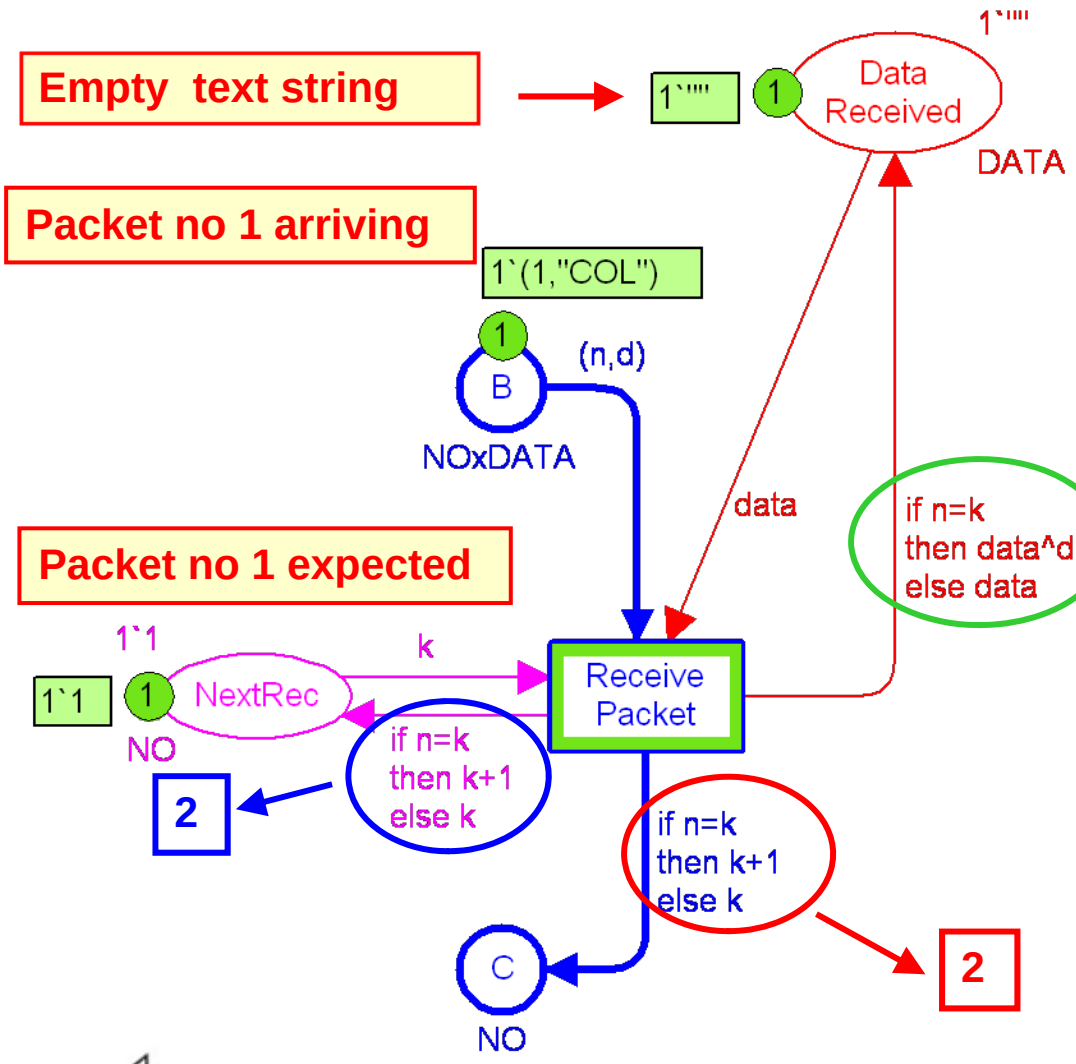
New Name and Type



New Place: NextRec



Correct packet arrives



Binding:
<n=1, d="COL“, k=1, data="">

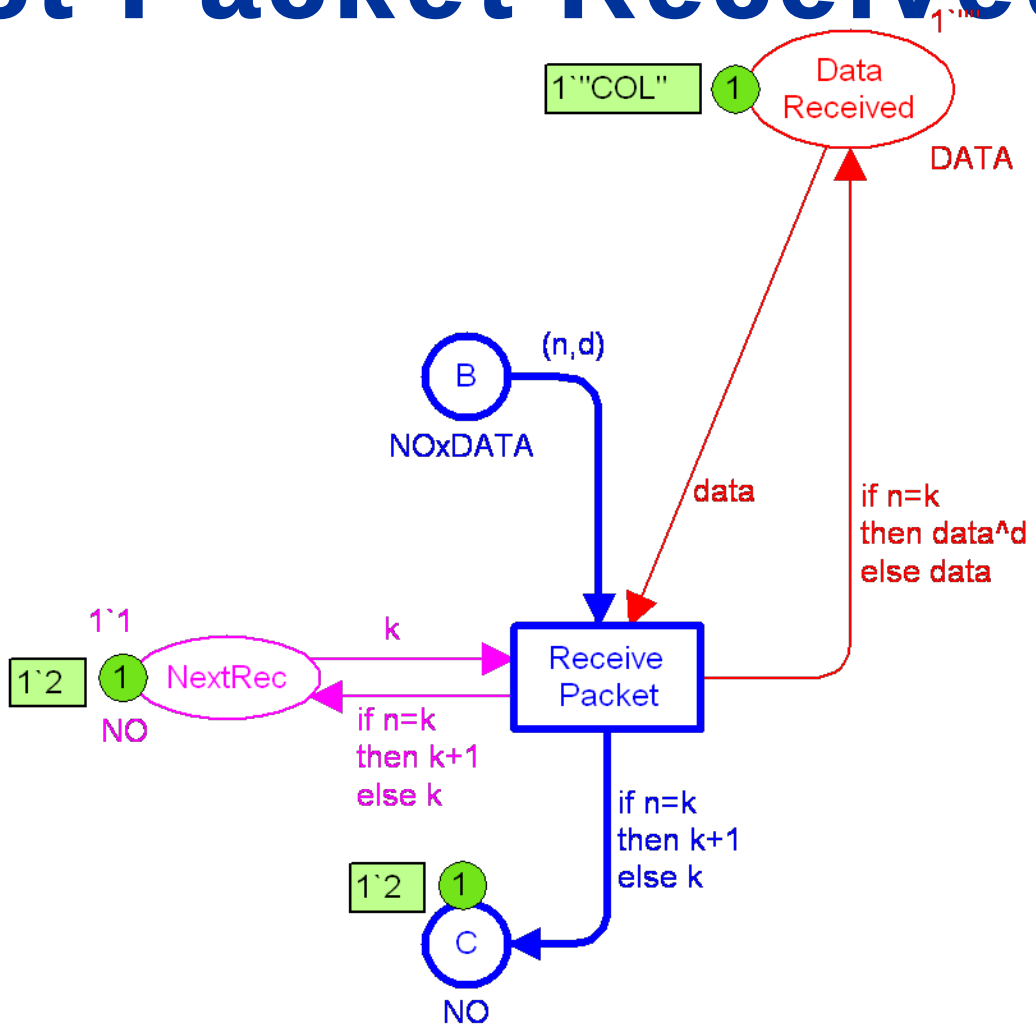
^ is the string concatenation operator

Update NextRec (from 1 to 2)

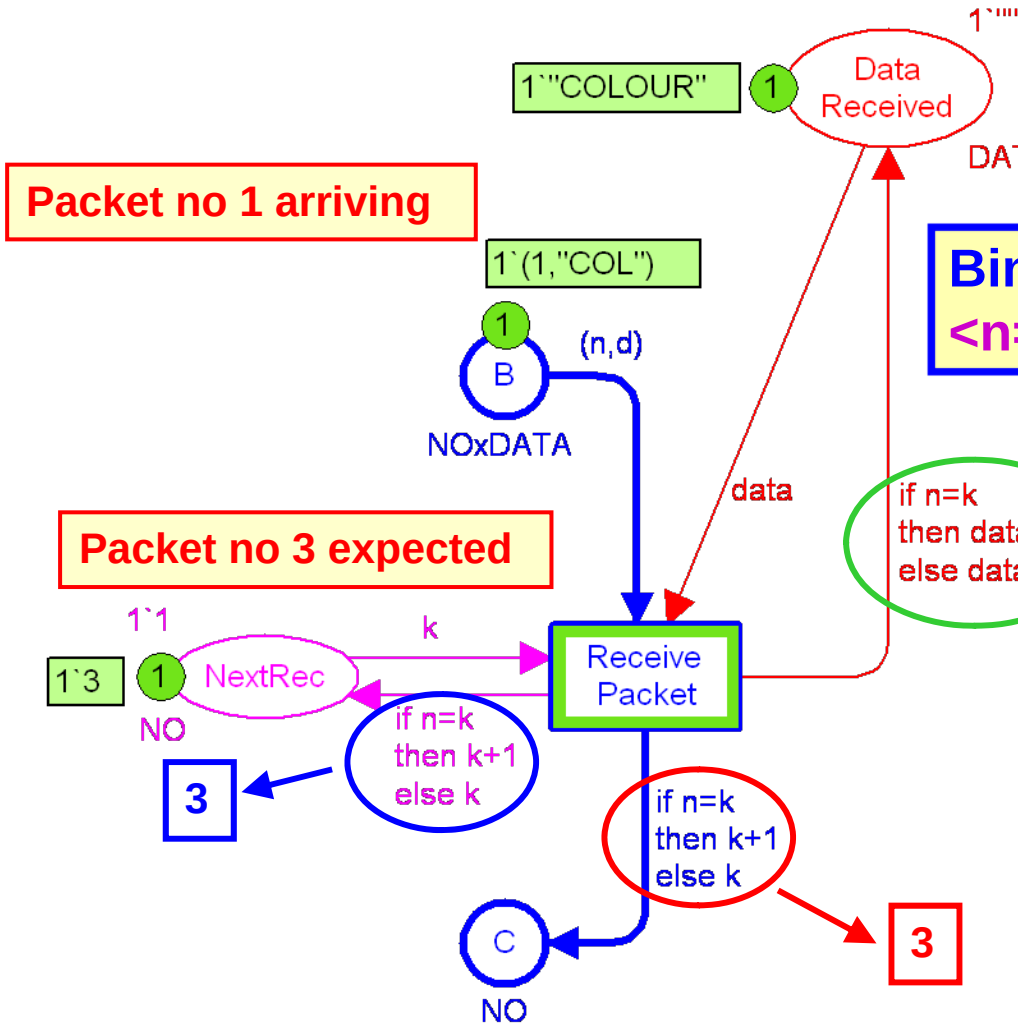
Send acknoweledgement (with sequence number of next packet)

Add received data: "COL"

Correct Packet Received



Wrong Packet Arrives



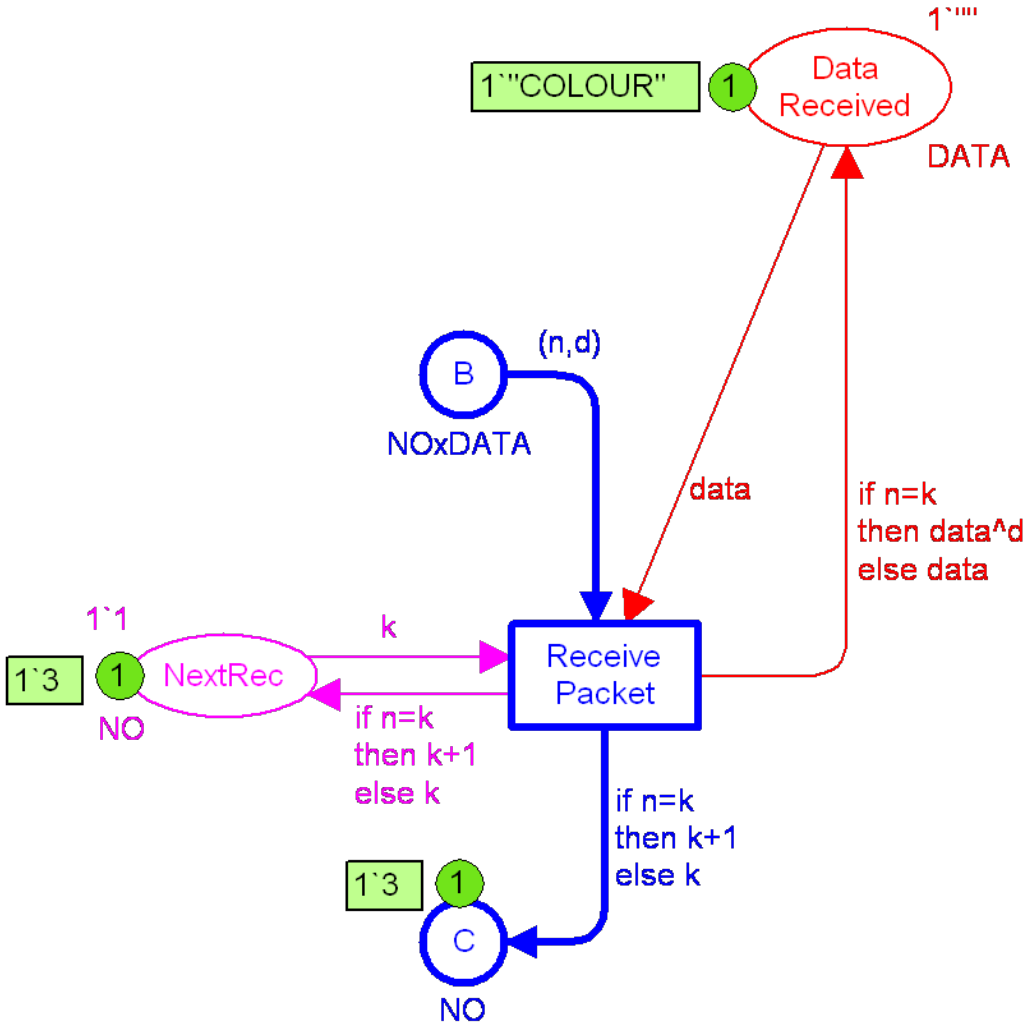
Binding:
<n=1, d="COL“, k=3, data="COLOUR">

Do not change NextRec

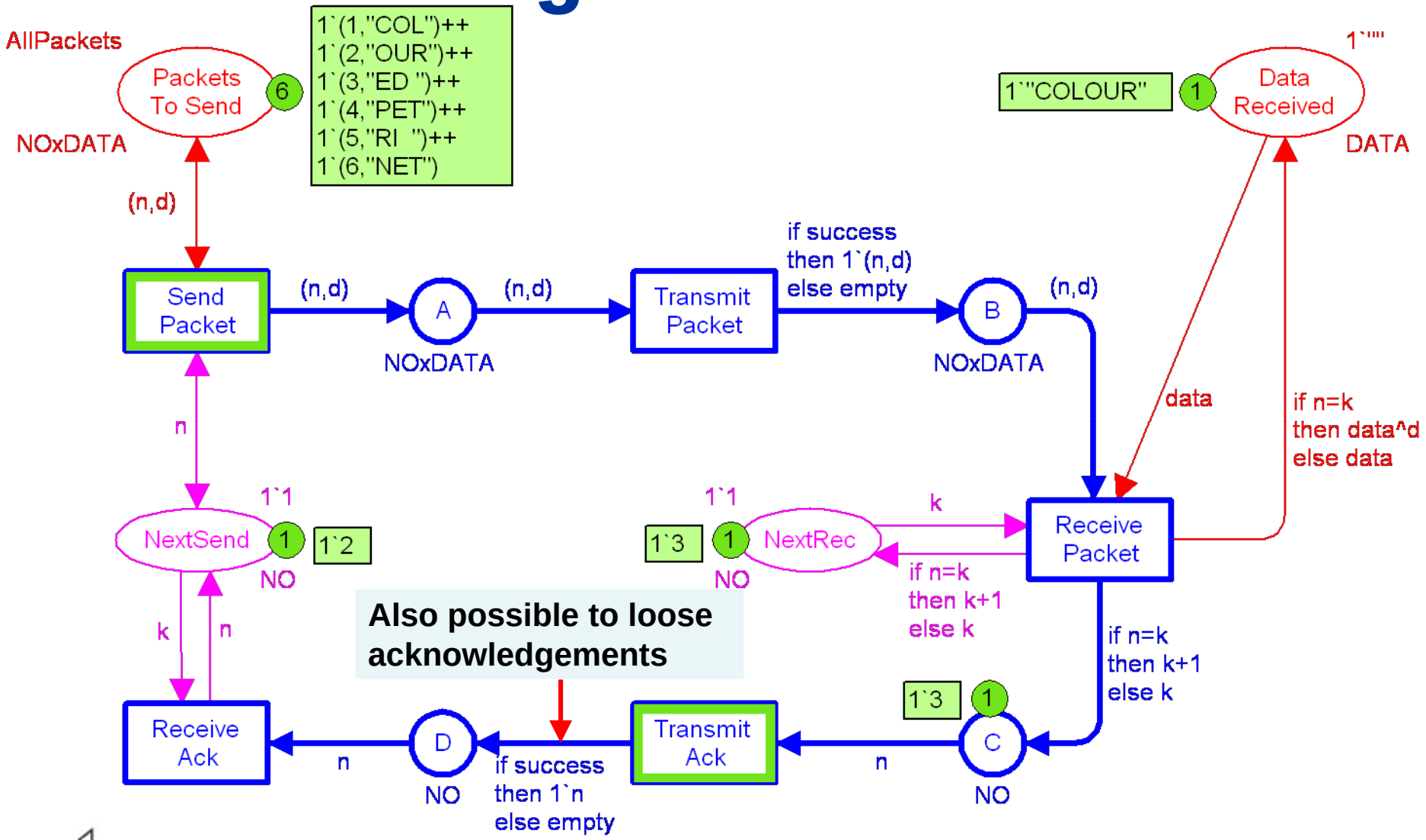
Send acknowledgement (with sequence number of next packet)

No data is added to DataReceived

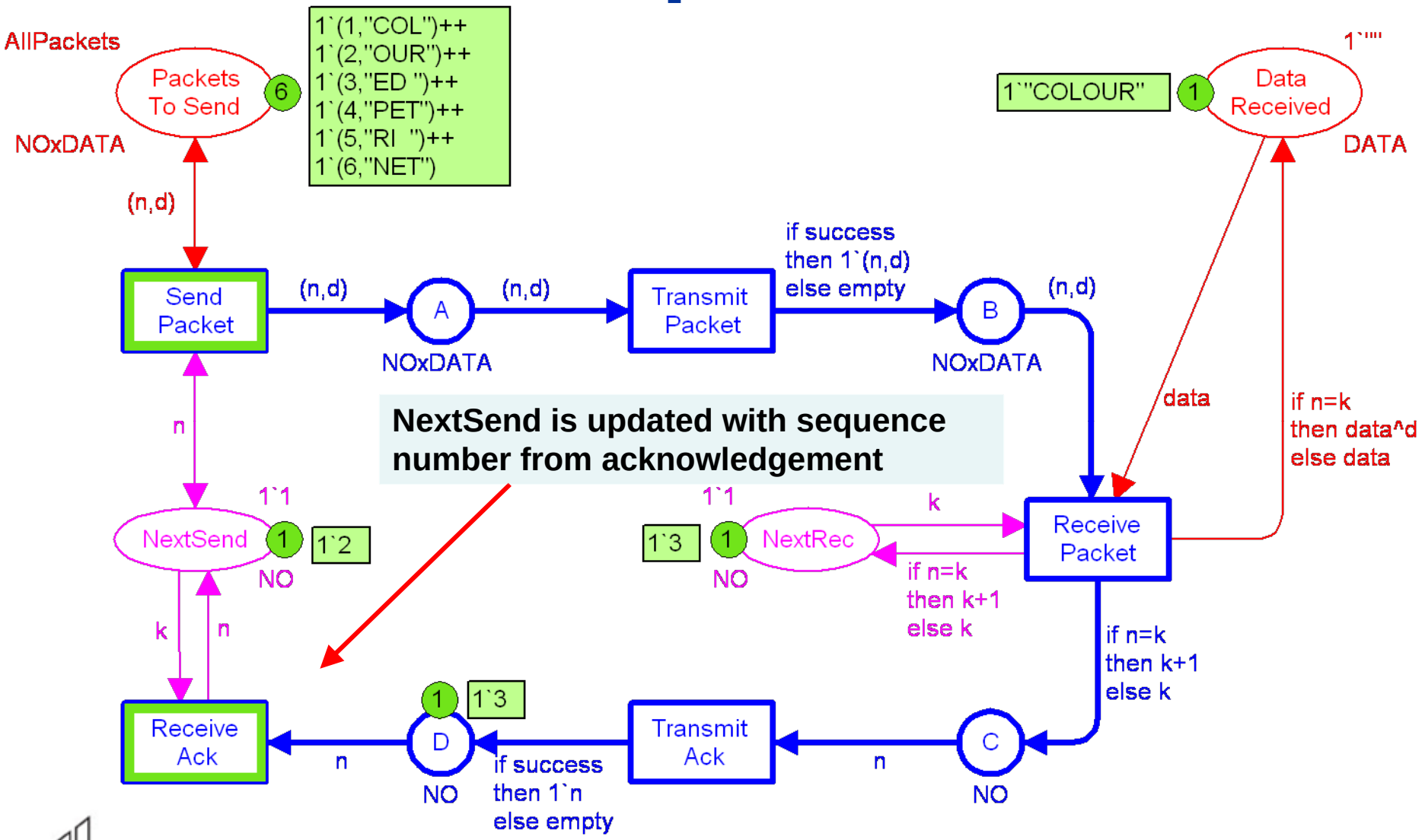
Wrong Packet Arrived

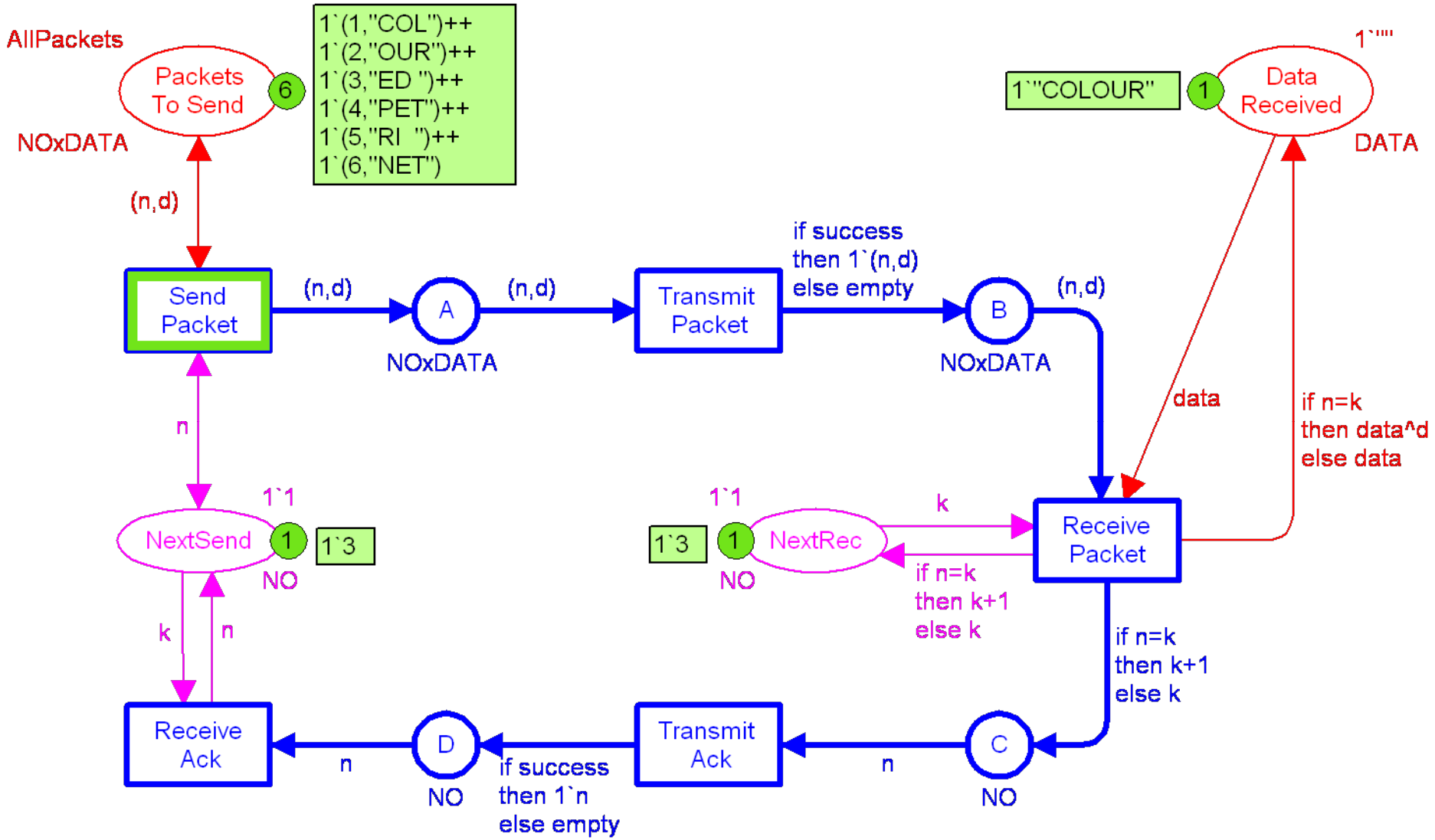


Acknowledgements can be lost



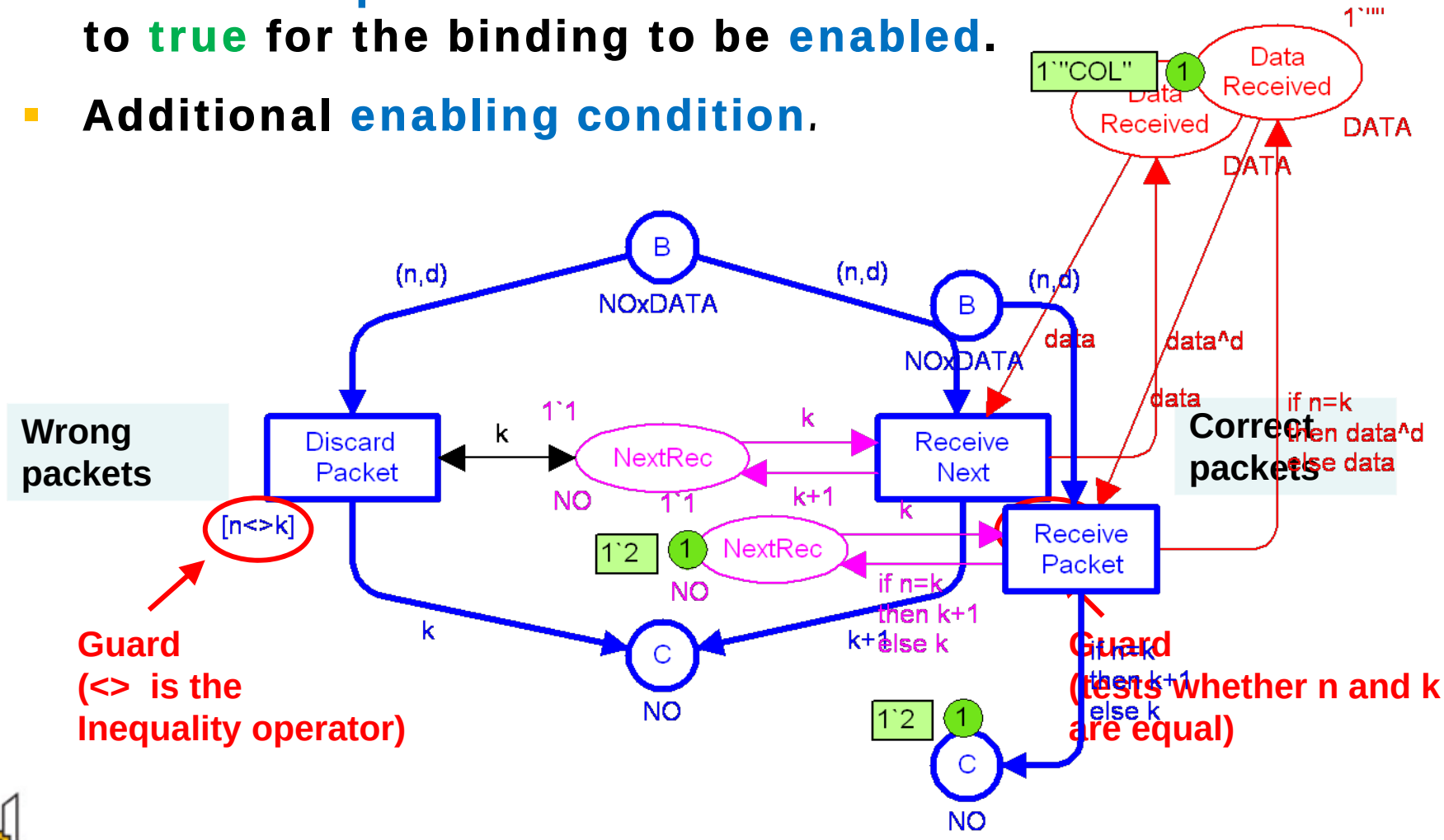
NextSend is updated



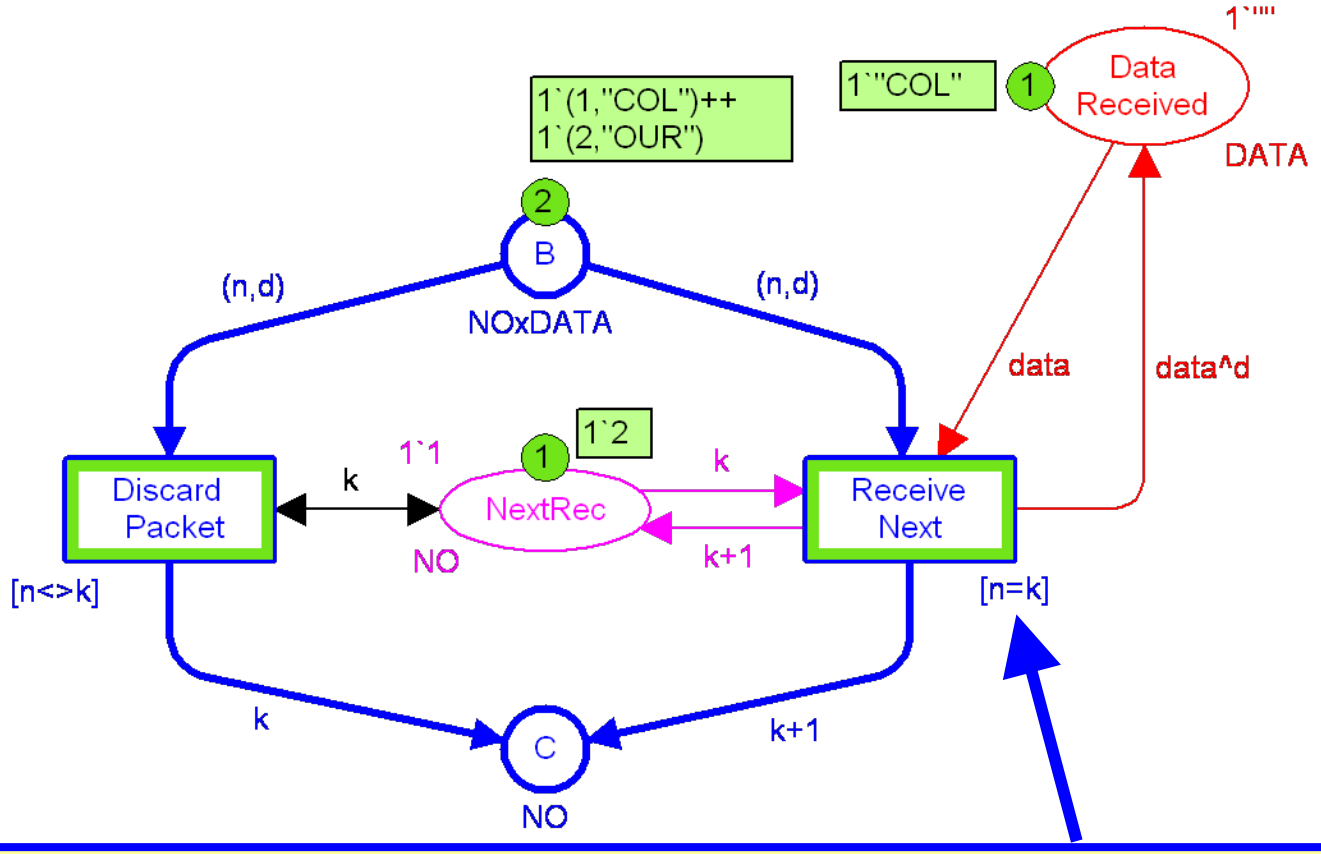

AllPackets
Packets To Send
6
1`(1,"COL")++
1`(2,"OUR")++
1`(3,"ED ")++
1`(4,"PET")++
1`(5,"RI ")++
1`(6,"NET")
NOxDATA
(n,d)
Send Packet
(n,d)
A
NOxDATA
(n,d)
Transmit Packet
if success
then 1`(n,d)
else empty
B
NOxDATA
(n,d)
1`"COLOUR"
1
Data Received
1`""
DATA
data
if n=k
then data^d
else data
n
1`1
NextSend
1
1`3
NO
1`1
1`3
1
NextRec
NO
k
if n=k
then k+1
else k
Receive Packet
if n=k
then k+1
else k
k
n
Receive Ack
n
D
NO
if success
then 1`n
else empty
Transmit Ack
n
C
NO

Transitions can have a guard

- **Boolean expression which must evaluate to true for the binding to be enabled.**
- **Additional enabling condition.**



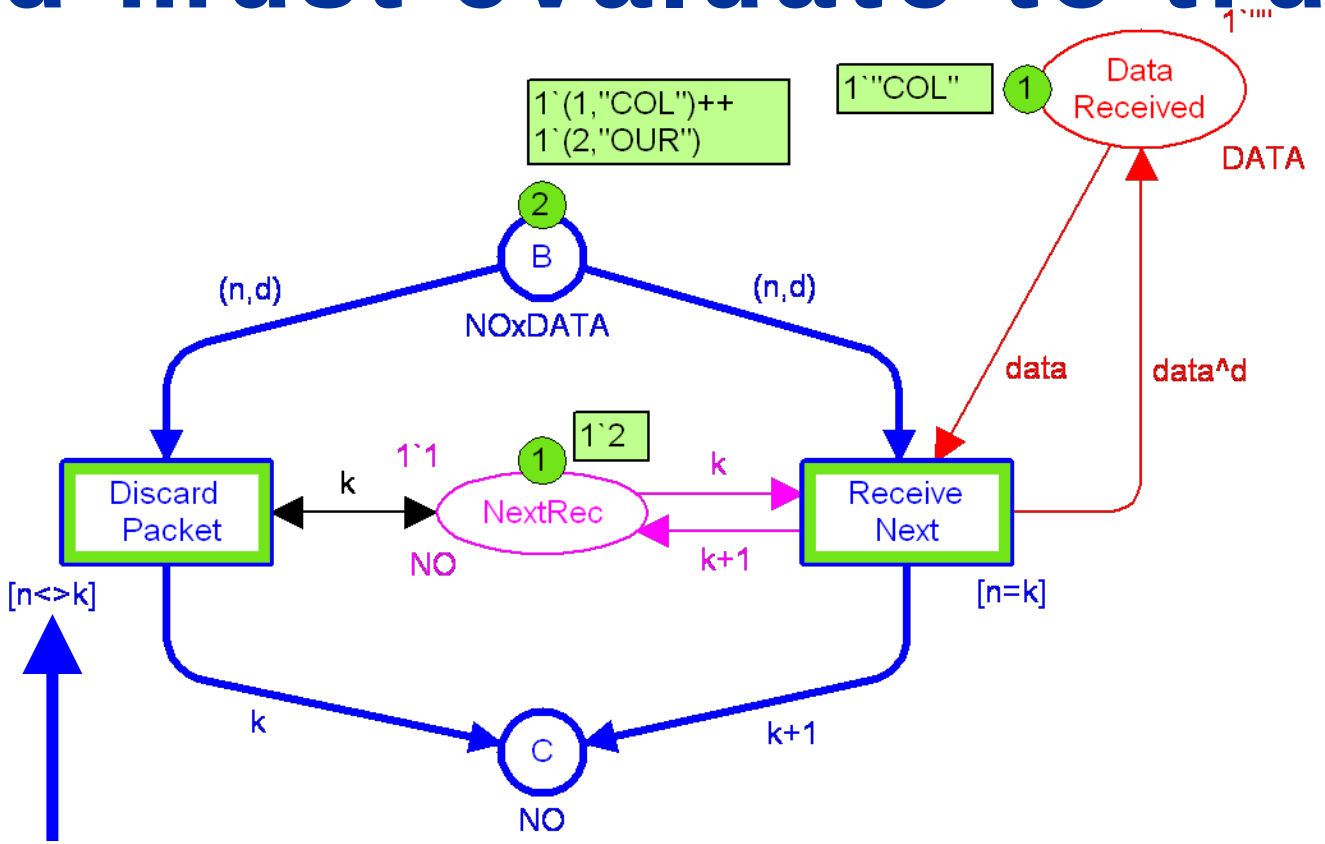
Guard must evaluate to true



false
true

RN_1 = (ReceiveNext, <n=1, k=2, d="COL", data="COL">)
RN_2 = (ReceiveNext, <n=2, k=2, d="OUR", data="COL">)

Guard must evaluate to true



true DP_1 = (DiscardPacket, <n=1, k=2, d="COL")

false DP_2 = (DiscardPacket, <n=2, k=2, d="OUR")

Formal Definition of CPNs

Definition 4.2. A **non-hierarchical Coloured Petri Net** is a nine-tuple $CPN = (P, T, A, \Sigma, V, C, G, E, I)$, where:

1. P is a finite set of **places**.
2. T is a finite set of **transitions** T such that $P \cap T = \emptyset$.
3. $A \subseteq P \times T \cup T \times P$ is a set of directed **arcs**.

Net structure

4. Σ is a finite set of non-empty **colour sets**.
5. V is a finite set of **typed variables** such that $Type[v] \in \Sigma$ for all variables $v \in V$.

Types and variables

6. $C : P \rightarrow \Sigma$ is a **colour set function** that assigns a colour set to each place.
7. $G : T \rightarrow EXPR_V$ is a **guard function** that assigns a guard to each transition t such that $Type[G(t)] = Bool$.
8. $E : A \rightarrow EXPR_V$ is an **arc expression function** that assigns an arc expression to each arc a such that $Type[E(a)] = C(p)_{MS}$, where p is the place connected to the arc a.
9. $I : P \rightarrow EXPR_\emptyset$ is an **initialisation function** that assigns an initialisation expression to each place p such that $Type[I(p)] = C(p)_{MS}$.

Net inscriptions

□

Enabling and Occurrence

Definition 4.5. A step $Y \in BE_{MS}$ is **enabled** in a marking M if and only if the following two properties are satisfied:

1. $\forall (t,b) \in Y : G(t)\langle b \rangle$.
2. $\forall p \in P : \quad \overset{++}{\underset{MS}{\sum}}_{(t,b)\in Y} E(p,t)\langle b \rangle \ll= M(p)$.

When Y is enabled in M, it may **occur**, leading to the marking M' defined by:

3. $\forall p \in P : M'(p) = (M(p) -- \overset{++}{\underset{MS}{\sum}}_{(t,b)\in Y} E(p,t)\langle b \rangle) ++ \overset{++}{\underset{MS}{\sum}}_{(t,b)\in Y} E(t,p)\langle b \rangle$.

□

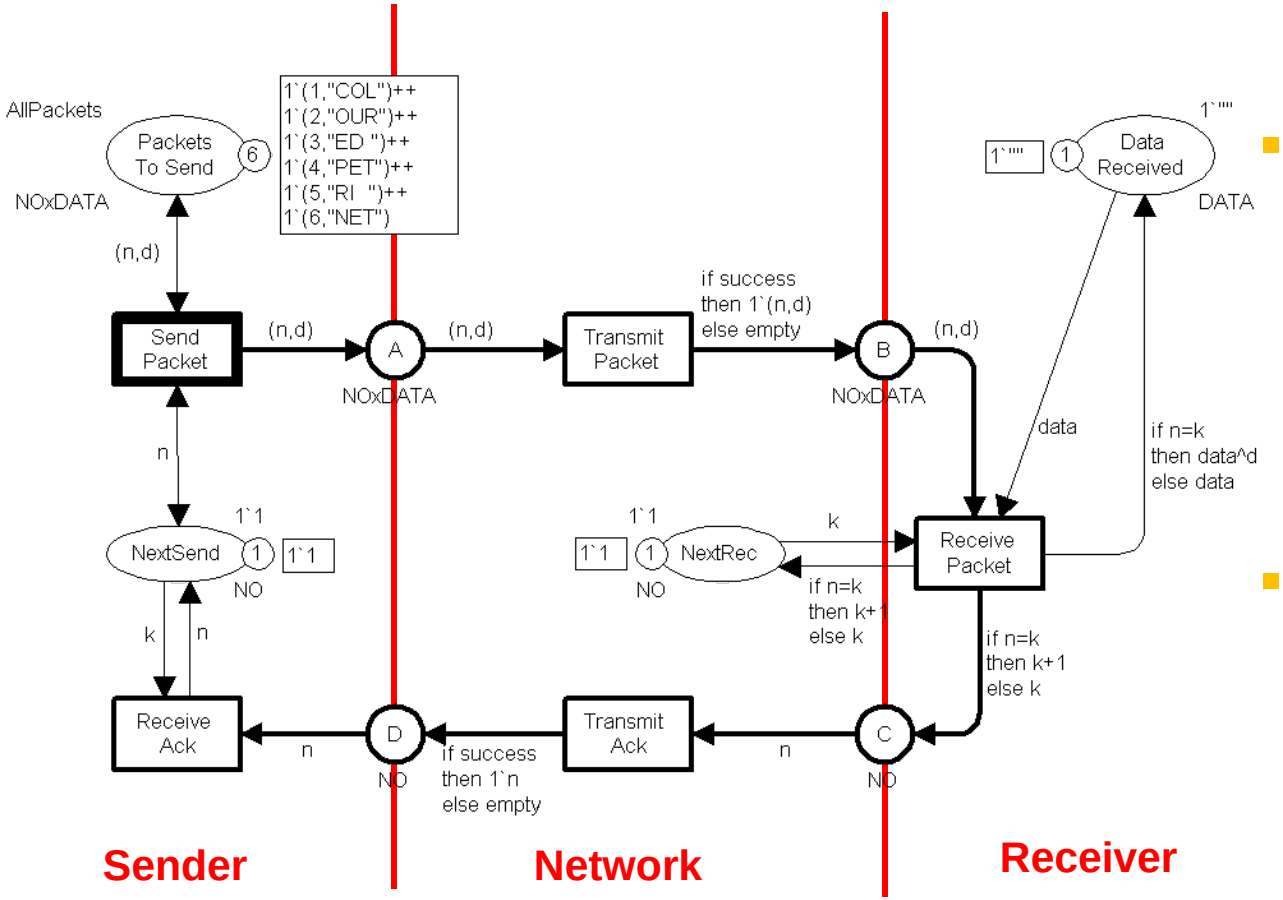
Part 3: Hierarchical Protocol CPN Model

CPN Modules

- CPN models can be hierarchically organised into a set of modules with well-defined interfaces:
  - Makes it possible to split models of large systems into manageable parts.
  - Makes it possible to work at different abstraction levels and have the model reflect the structure of the system.
  - Makes it possible to create building blocks that are used repeatedly in the CPN model.

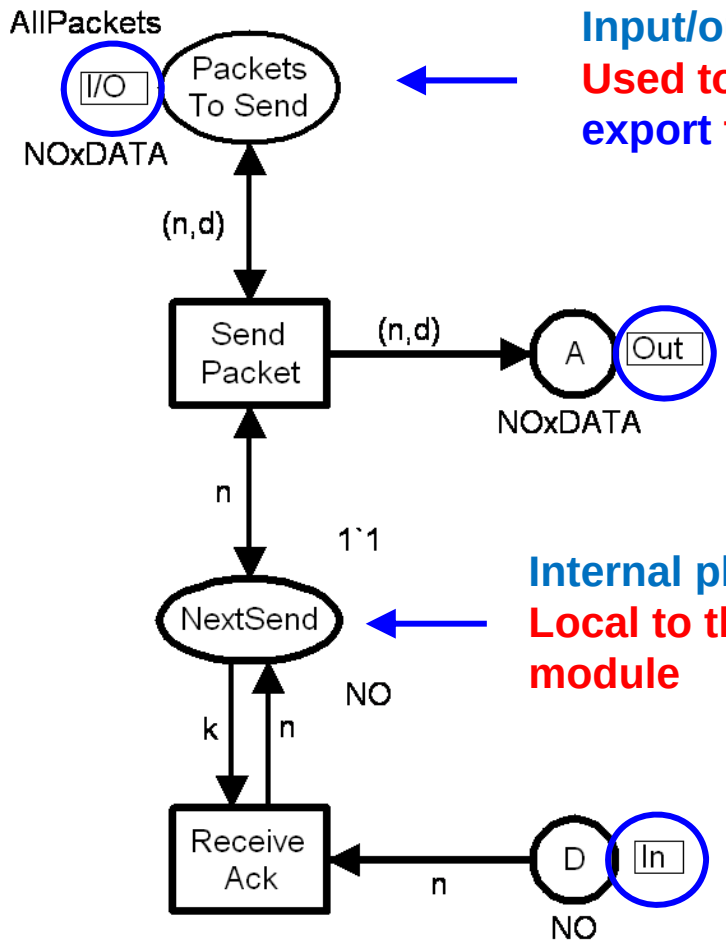
- CPN models of larger systems typically have up to 10 abstraction (hierarchical) levels.
- CPN models with modules are also called hierarchical Coloured Petri Nets.

Simple Protocol



- **The protocol model can be divided into three modules:**
  - **Sender**
  - **Network**
  - **Receiver**
- **The buffer places are used as interfaces between the modules.**

Sender Module



Input/output port
Used to import and export tokens

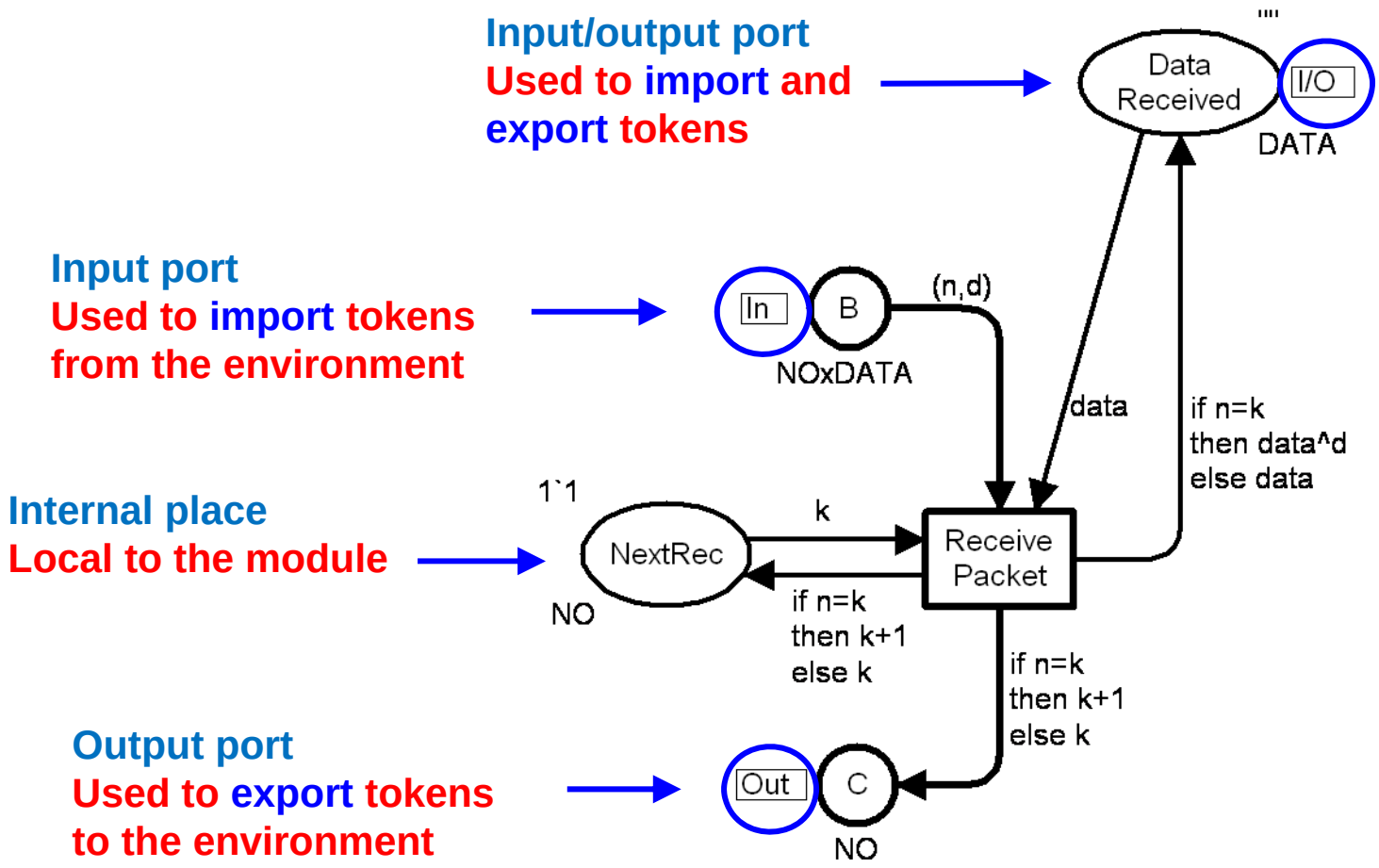
Output port
Used to export tokens to the environment

Internal place
Local to the module

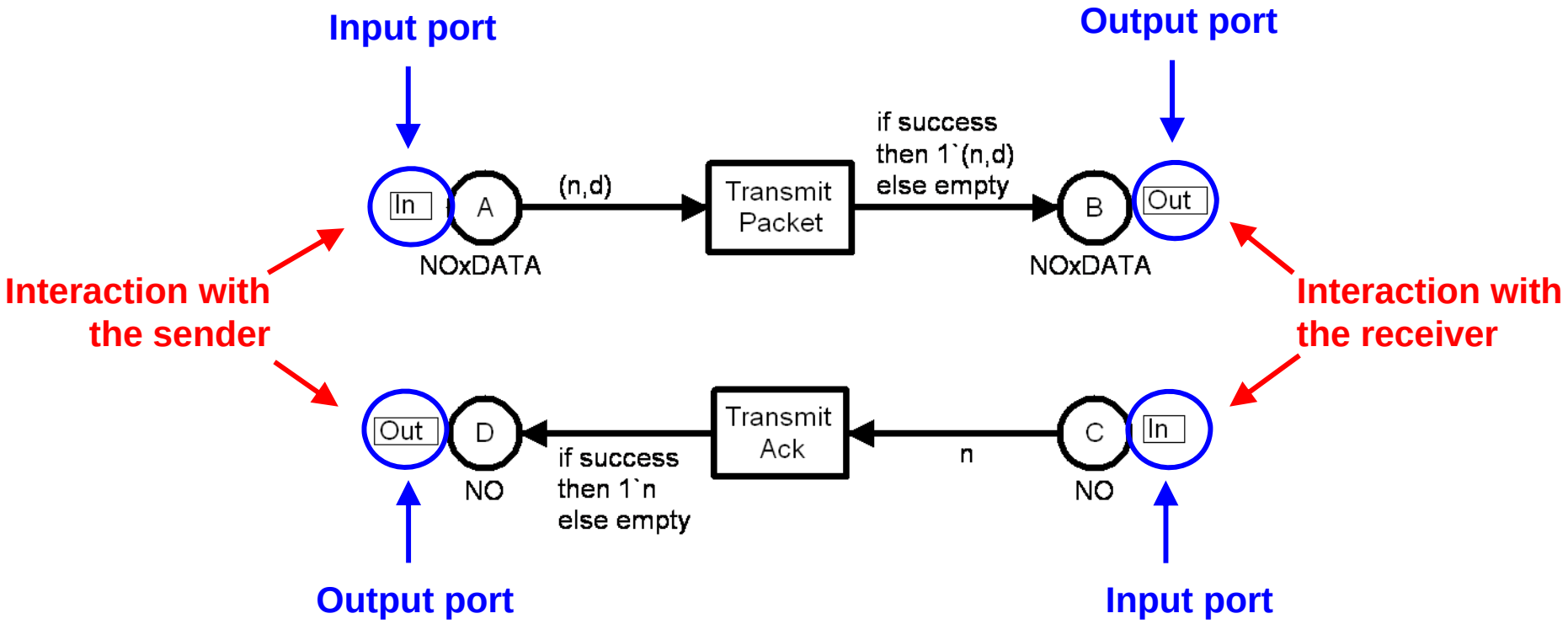
The module interacts with other modules via three port places.

Input port
Used to import tokens from the environment

Receiver module

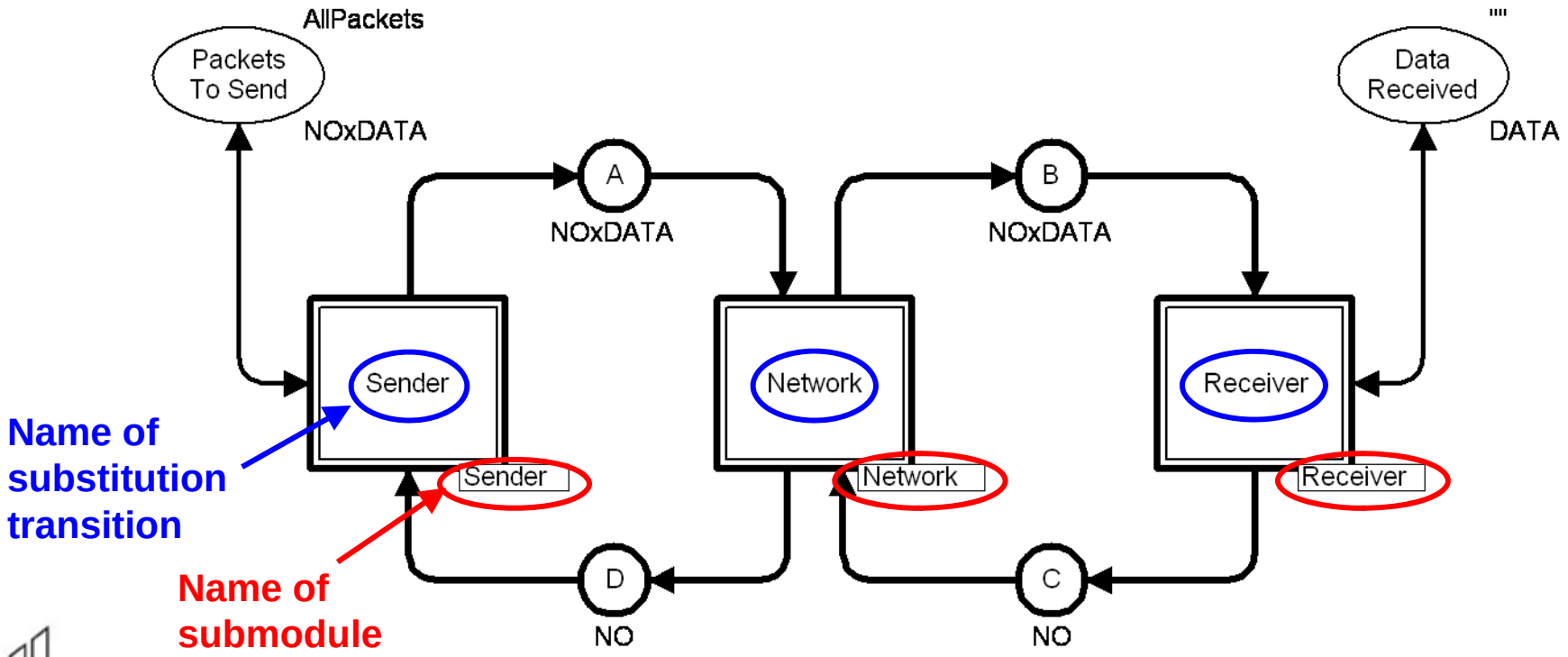


Network module

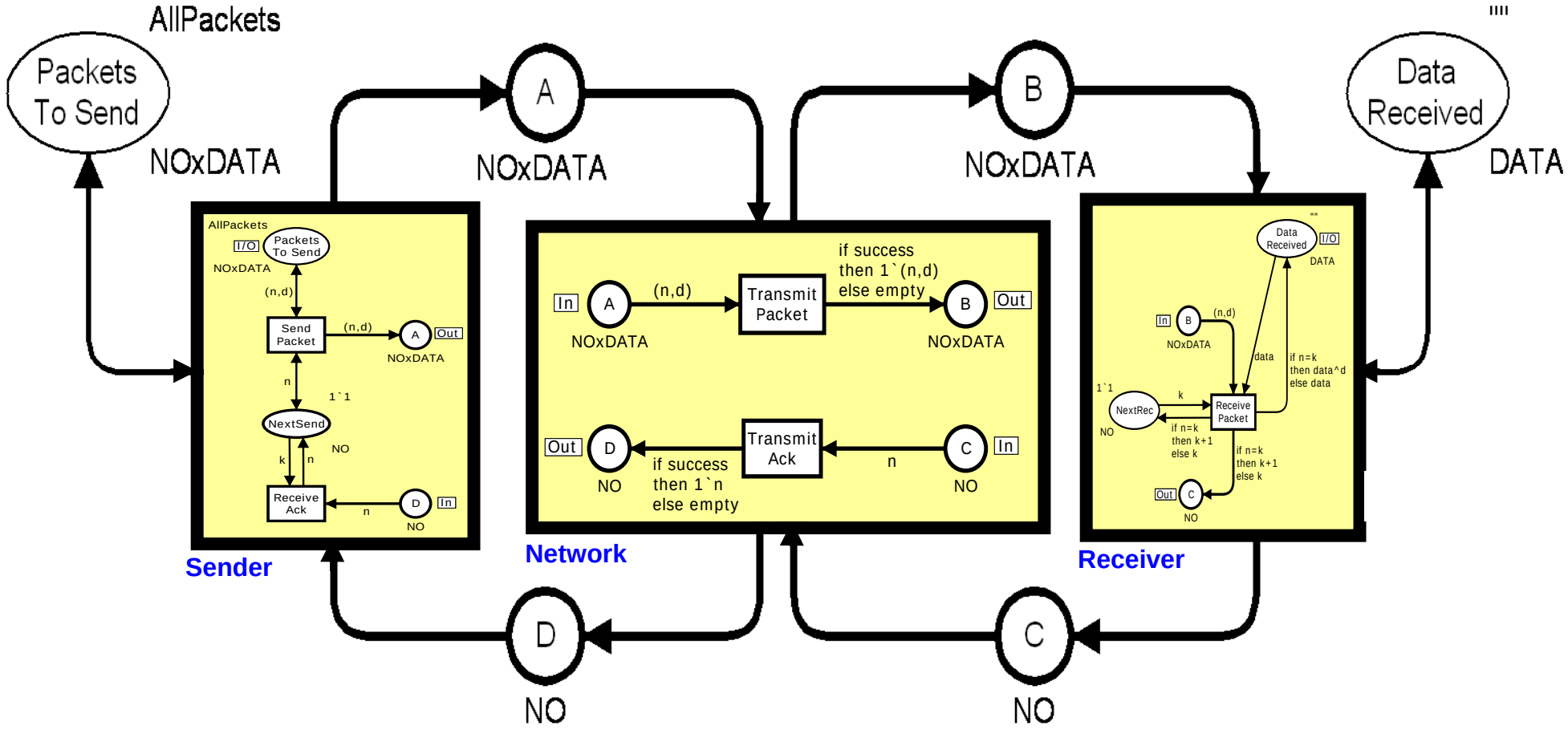


Protocol Module

- Ties the three other modules together using **substitution transitions**.
- Provides a more **abstract view** of the protocol system.

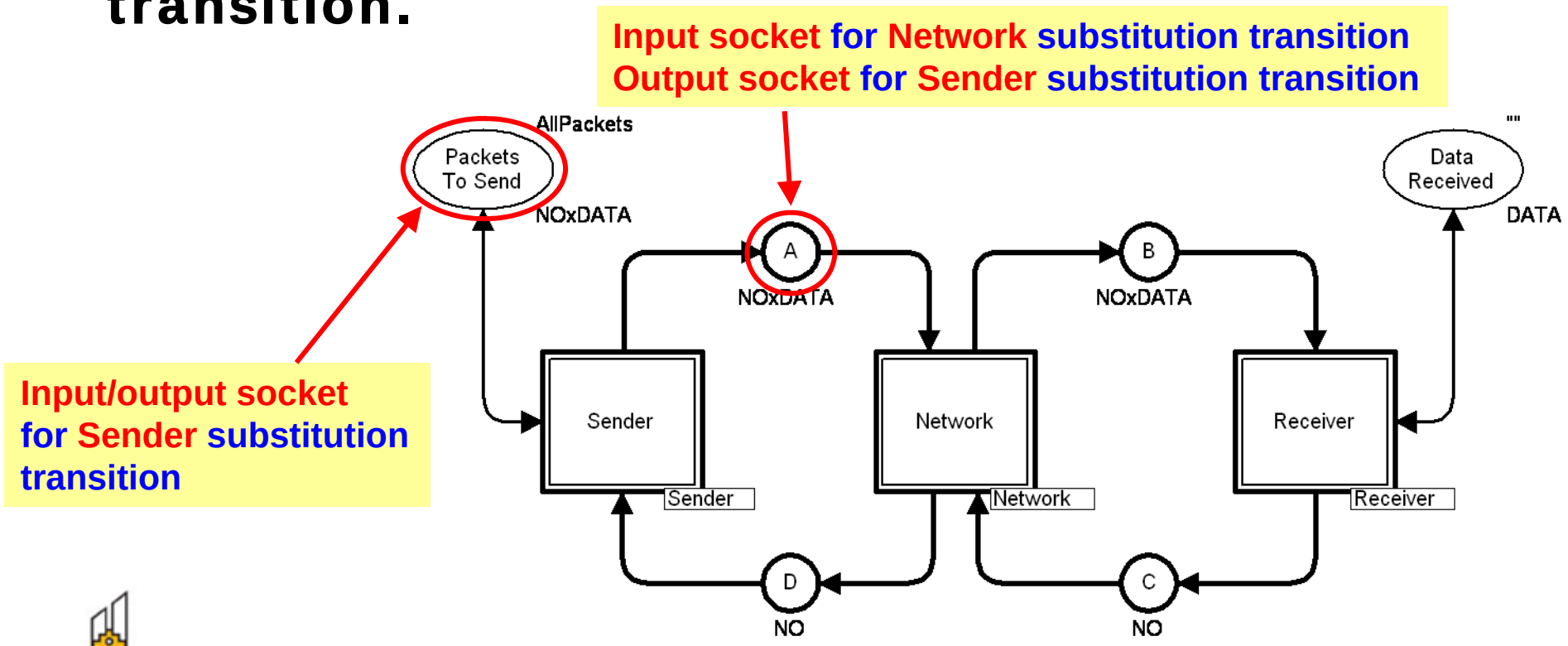


Protocol Module



Protocol Module

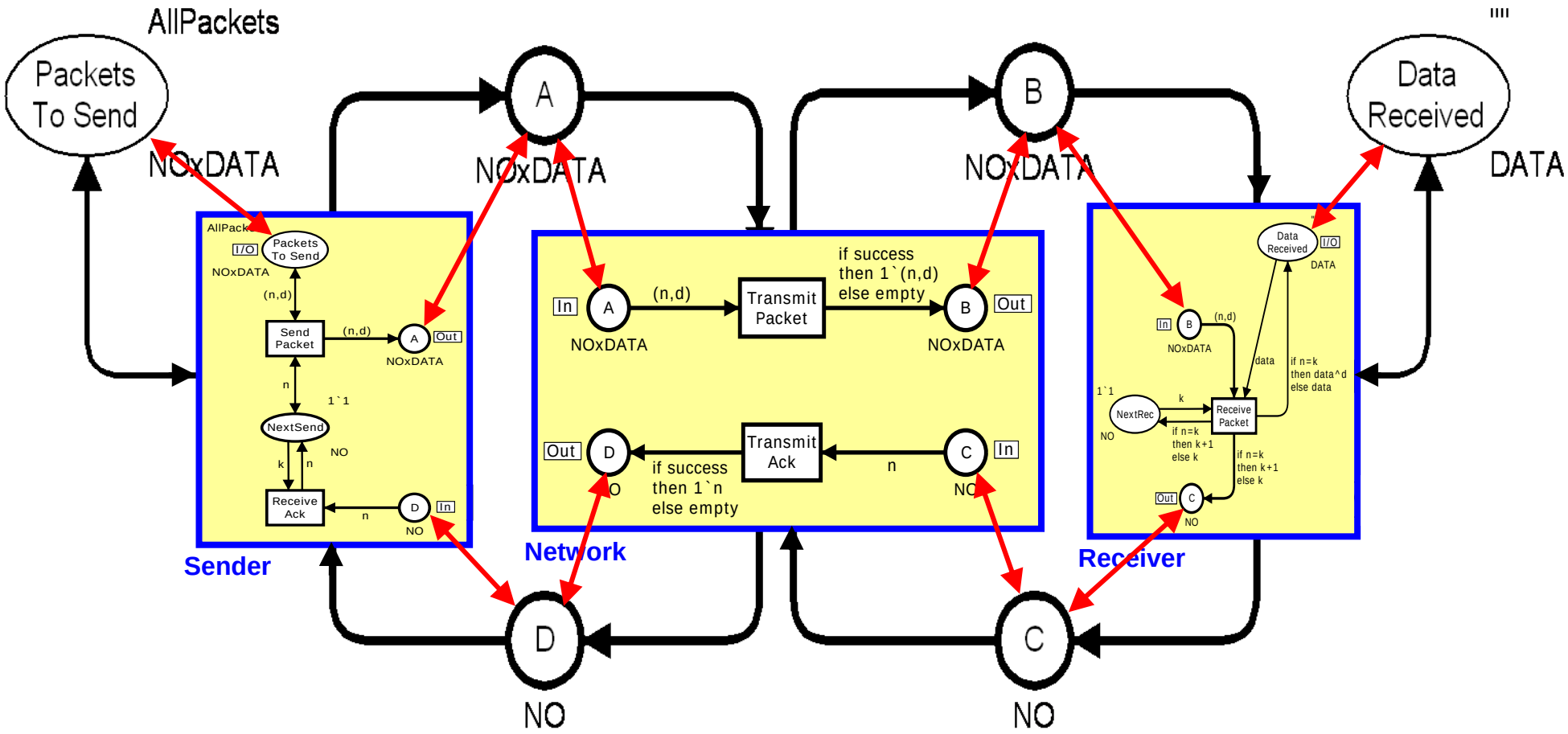
- The places connected to the substitution transitions are socket places.
- They constitute the interface for the substitution transition.



Port-Socket Relation

- **Each port place of a submodule is related to a socket place of its substitution transition:**
  - input port ↔ input socket.
  - output port ↔ output socket.
  - input/output port ↔ input/output socket.
- **Ports and sockets that are related to each other constitute a single compound place.**
  - **They have the same marking.**
  - **When a token is added / removed at one of them it is also added / removed at the other.**
  - **Also the colour sets and initial markings are required to be identical.**

Port-Socket Relation



For the protocol system ports and sockets have identical names, but this is not required in general.

References: Getting Started

- K. Jensen, L.M. Kristensen, and L. Wells. Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. In International Journal on Software Tools for Technology Transfer (STTT), Vol 9, No. 3-4, pp. 213-254. Springer-Verlag, 2007.
- CPN Tools:
  [www.daimi.au.dk/CPNtools]
  (see Download and Installation section)
- K. Jensen and L.M. Kristensen.
  Coloured Petri Nets: Modelling and Validation of Concurrent Systems. Springer, 2009.
  [book web: www.cs.au.dk/CPnets/cpnbook/]