Coloured Petri Nets

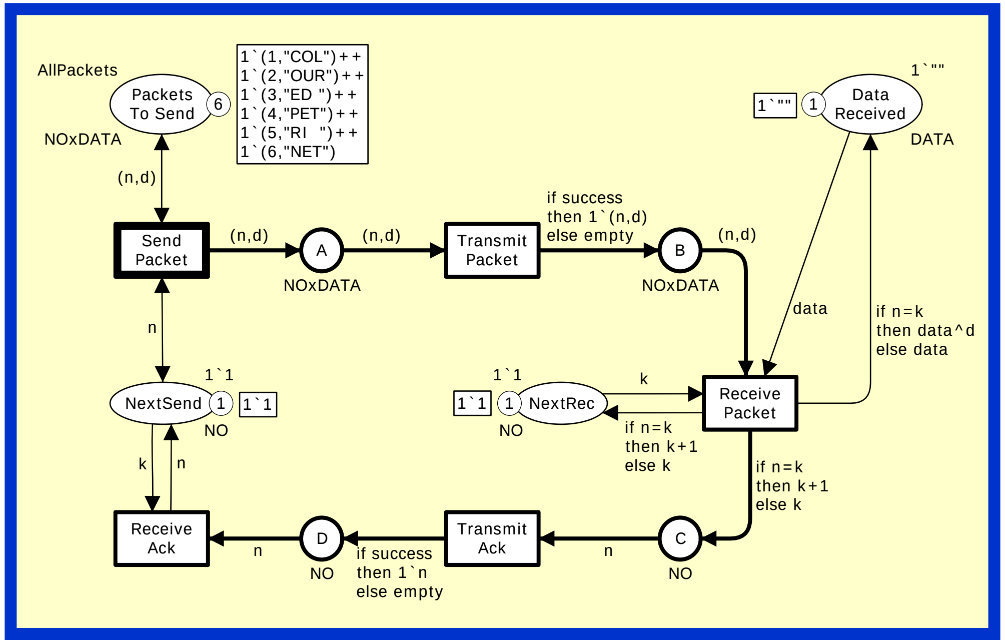
Modelling and Validation of Concurrent Systems

Chapter 2: Basic Concepts

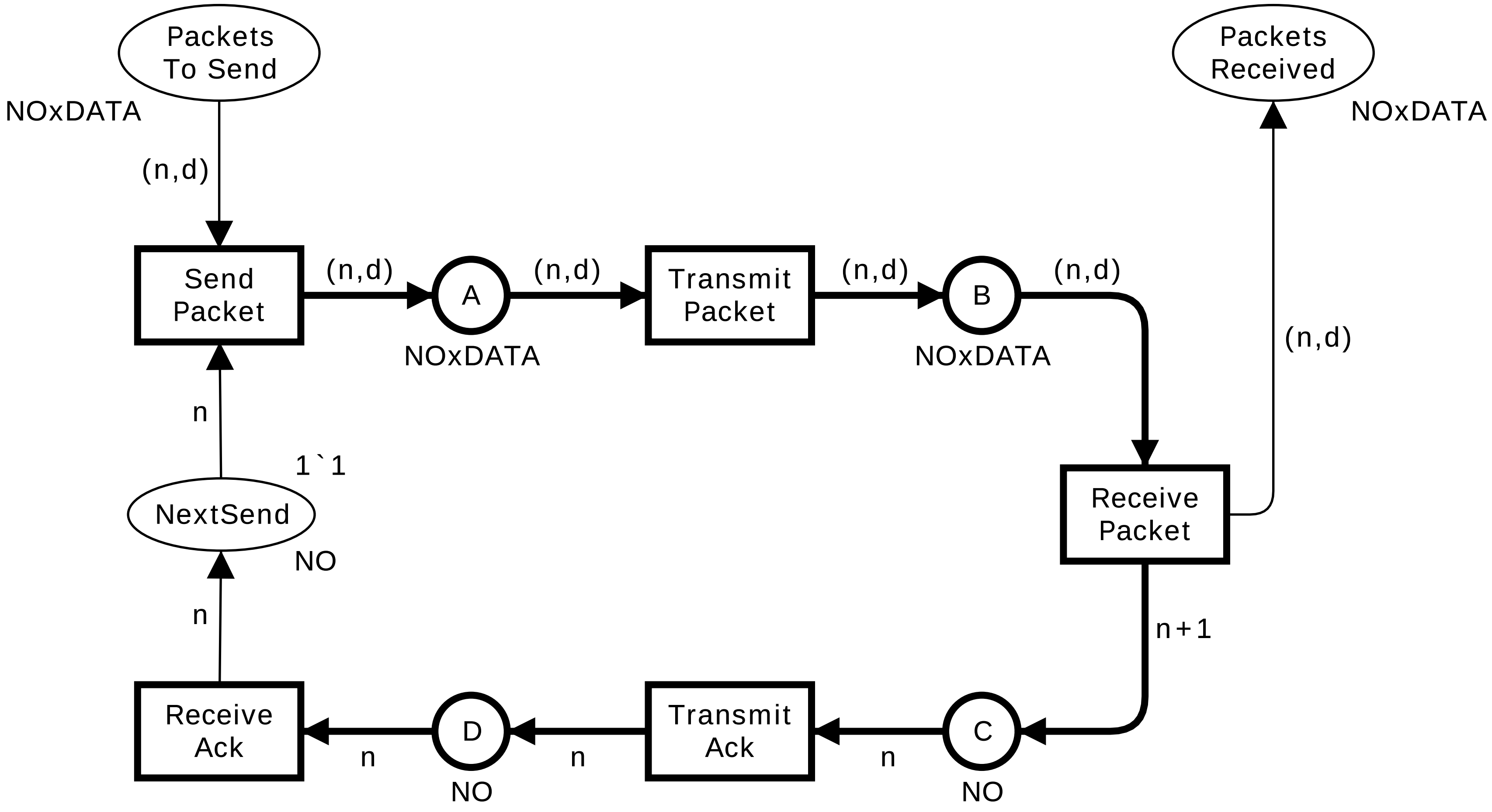
Kurt Jensen &
Lars Michael Kristensen

{kjensen,lmkristensen}
@daimi.au.dk

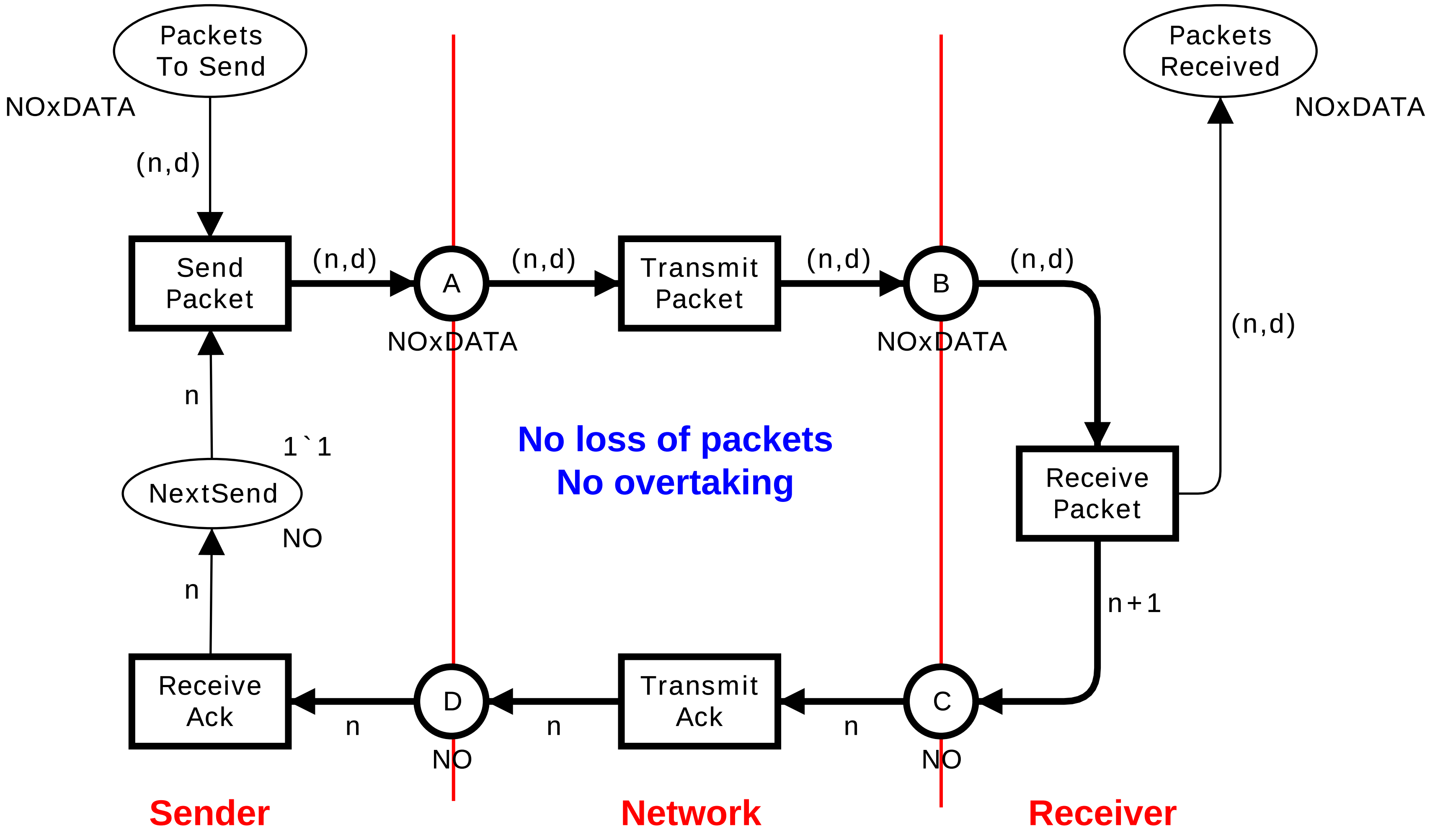
© January 2008



Simple protocol

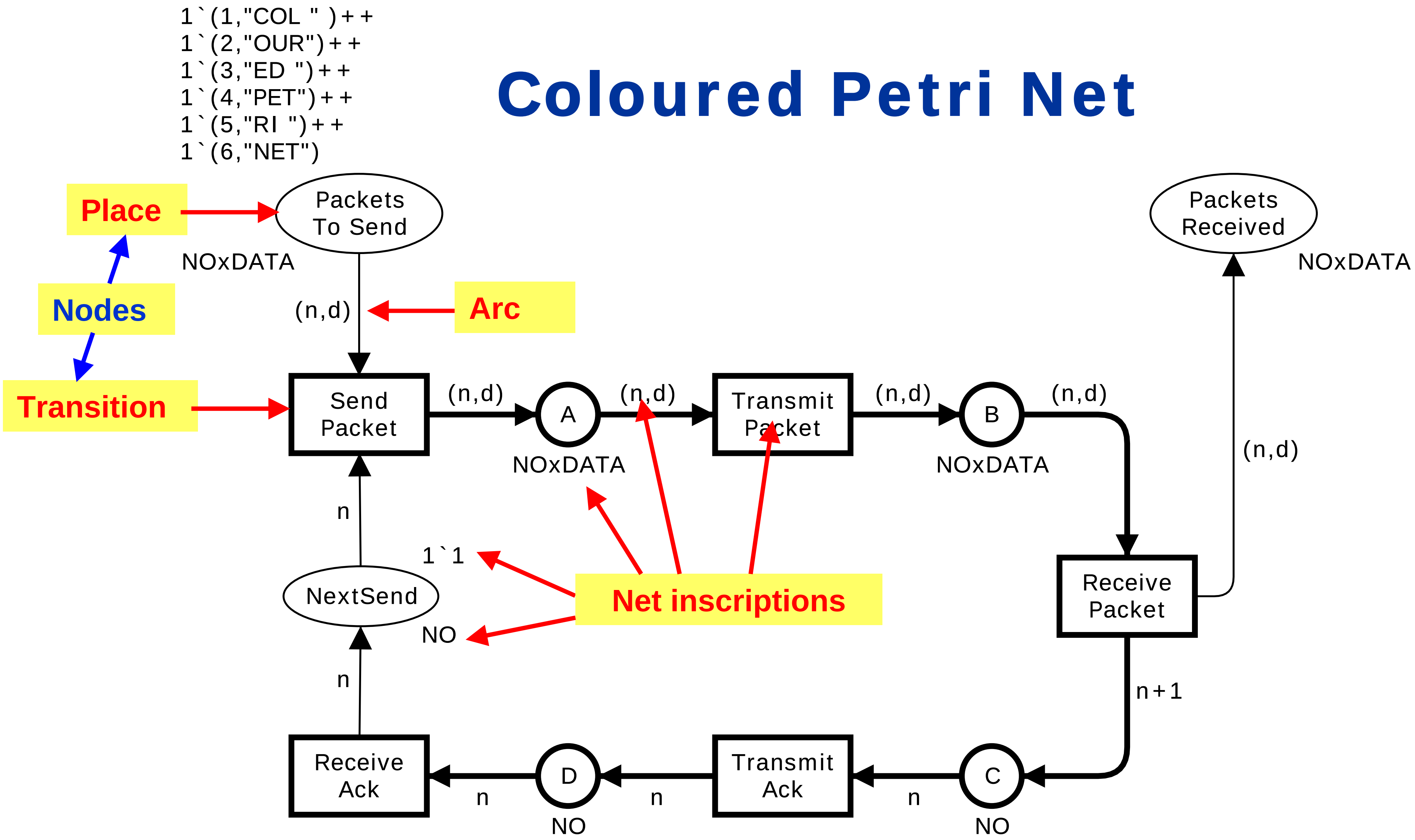


Informal description

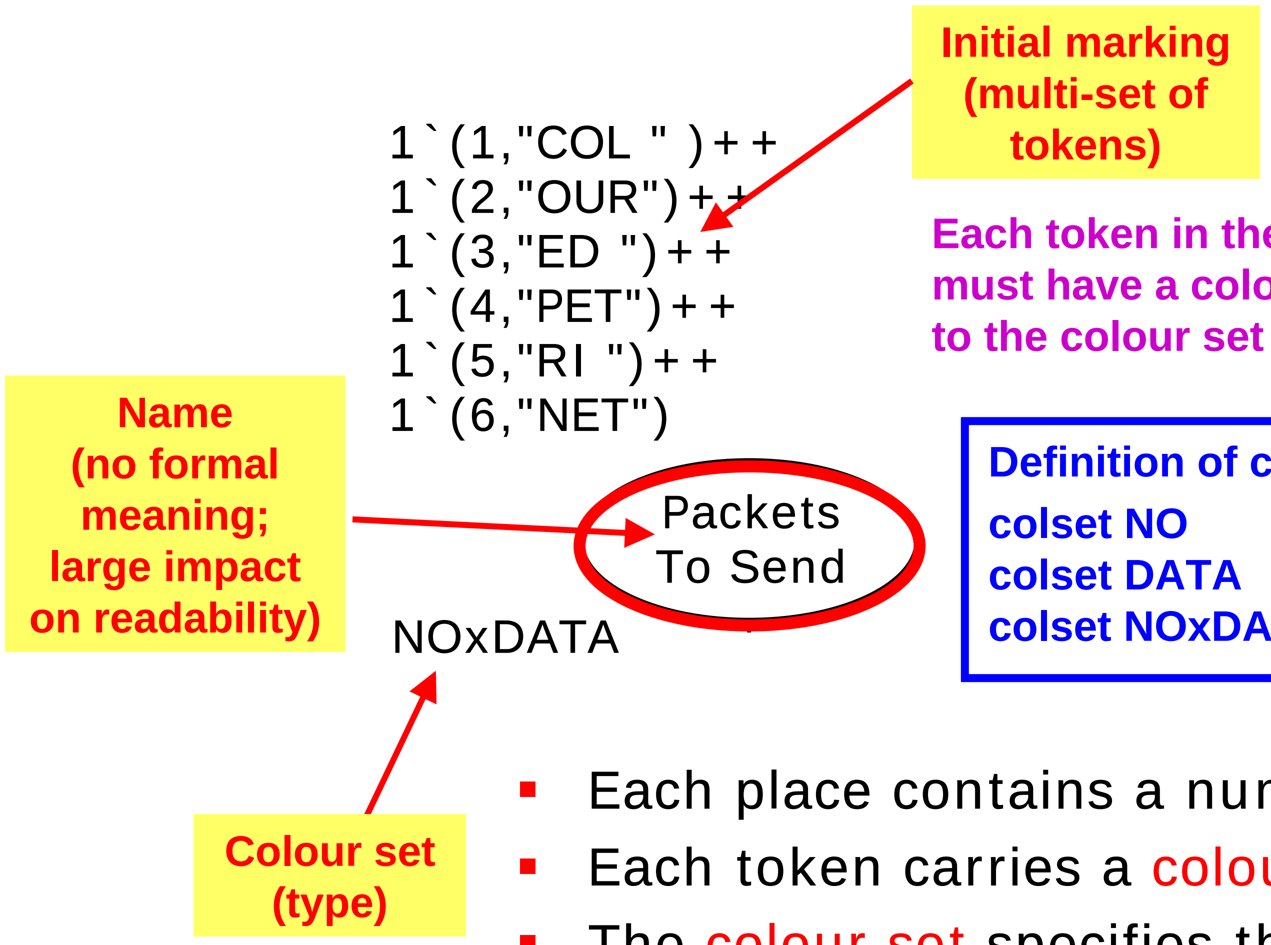


No loss of packets
No overtaking

Coloured Petri Net



Places represent the state of the system



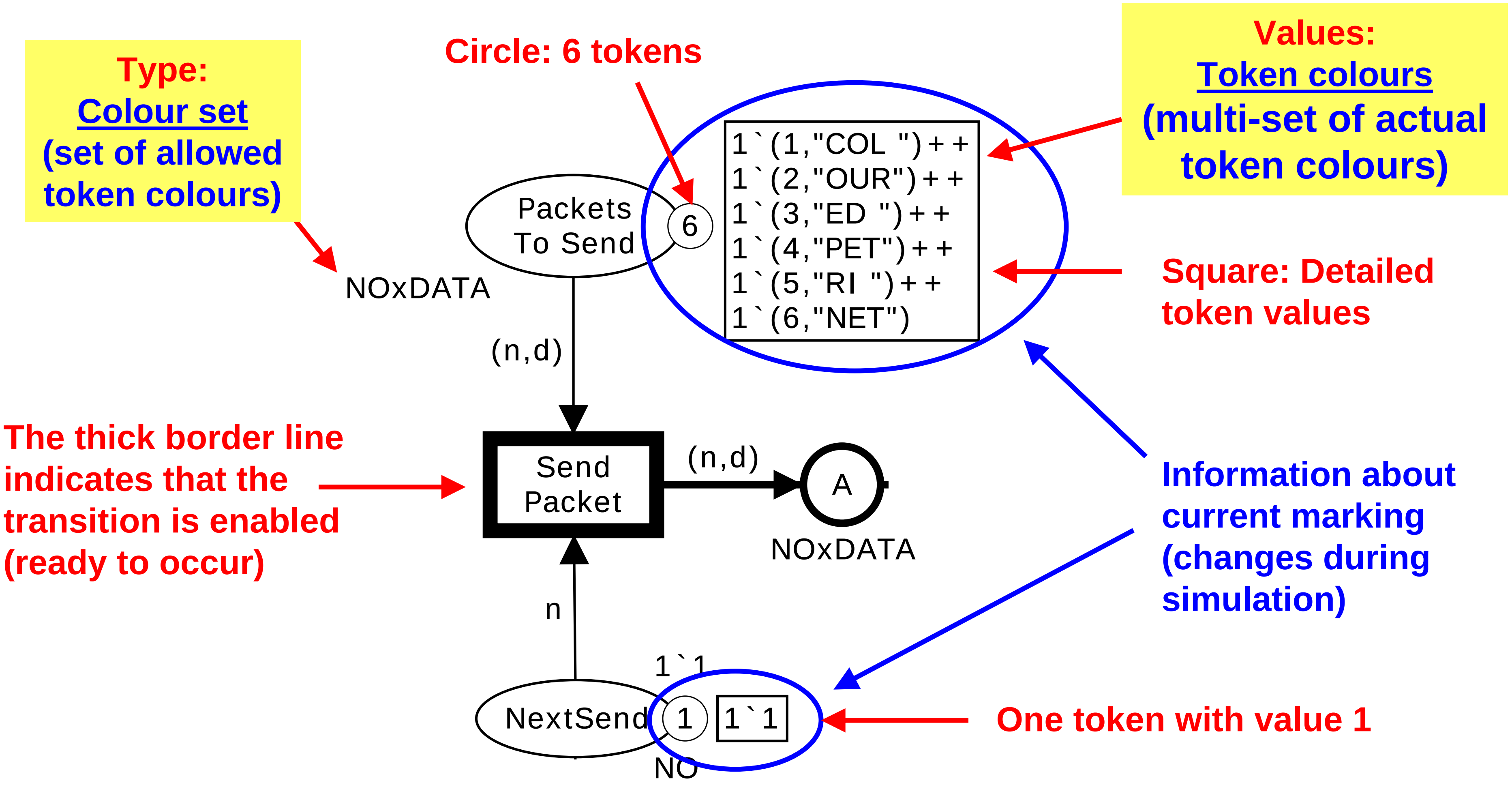
Each token in the initial marking must have a colour that belongs to the colour set

Definition of colour sets:

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

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

Current marking during simulation



Type:
Colour set
(set of allowed token colours)

Circle: 6 tokens

Values:
Token colours
(multi-set of actual token colours)

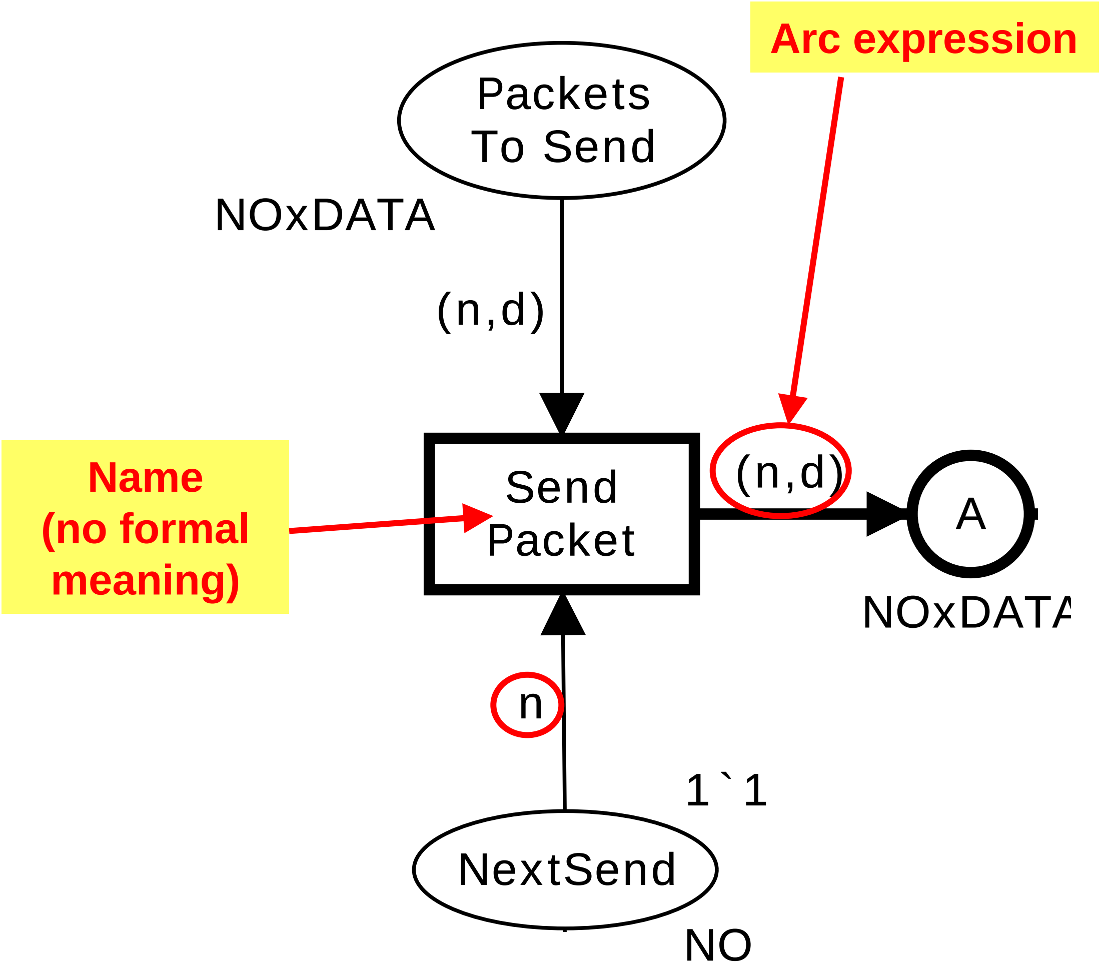
Square: Detailed token values

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

Information about current marking (changes during simulation)

One token with value 1

Transitions and arcs



The arc expression must evaluate to a colour in the colour set of the attached place
(or a multi-set of such colours)

Declaration of variables:

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

Binding of variables:

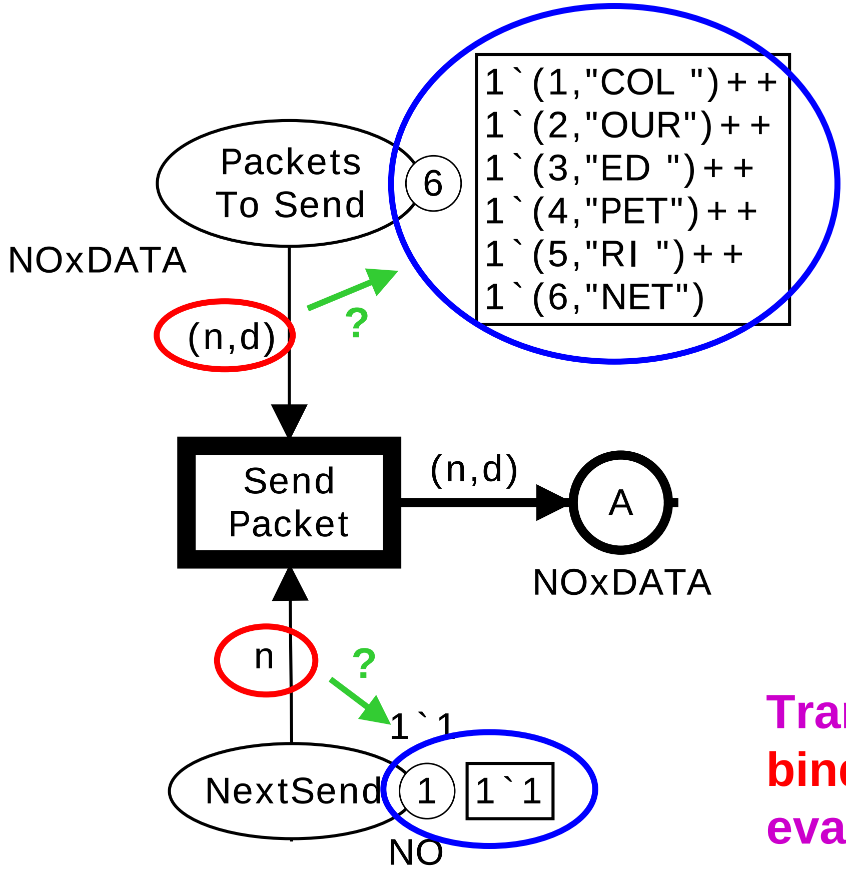
<n=3,d="CPN">

Evaluation of expressions:

(n,d) → (3,"CPN") : NOxDATA

n → 3 : NO

Enabling of transition



Two variables:

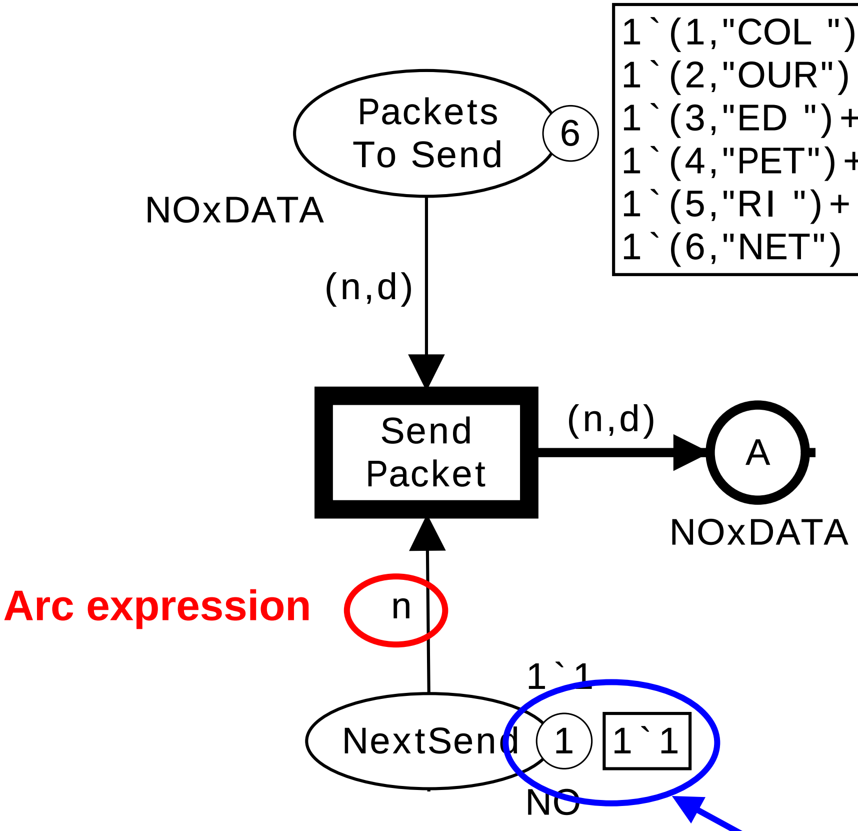
var n : NO; (* integers *)
var d : DATA; (* strings *)

Binding: < n=? , d=? >

NO DATA

Transition is enabled if we can find a binding so that each input arc expression evaluates to one or more colours that are present on the corresponding input place

Enabling of SendPacket

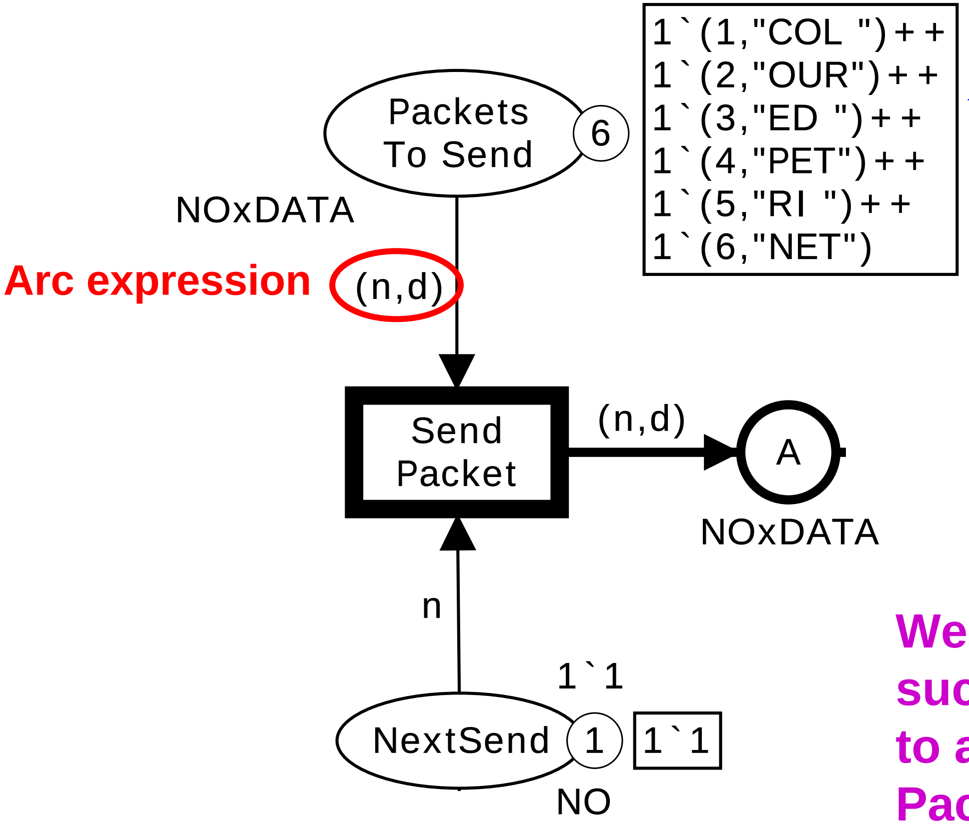


Binding: < n=1 , d=? >

We want to find a binding for the **variable n** such that the **arc expression n** evaluates to a **colour** which is present on the place NextSend

One token with value 1

Enabling of SendPacket



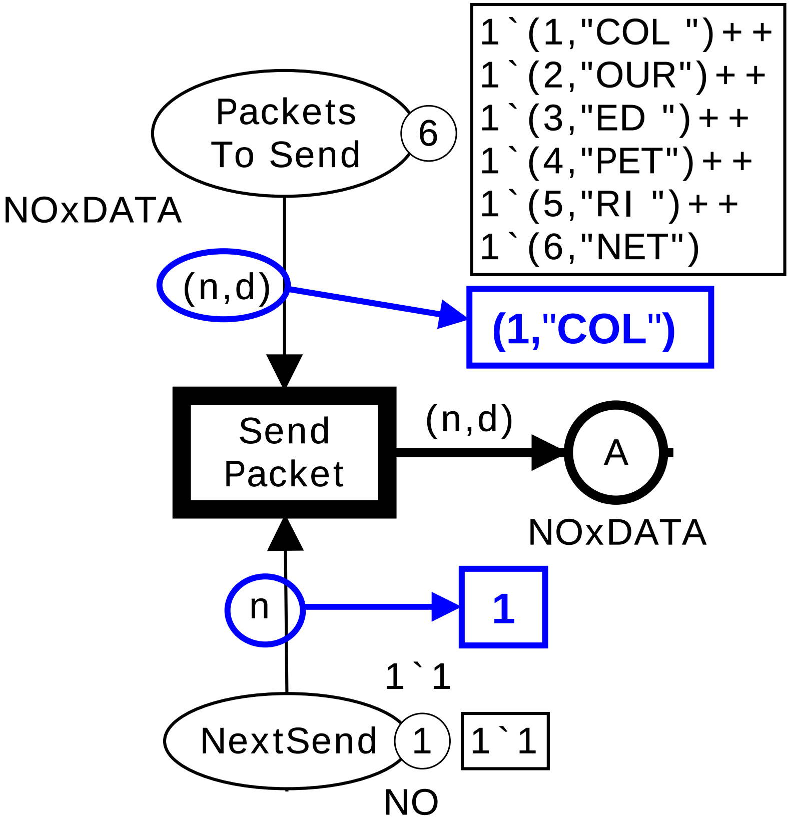
1`(1,"COL ")++
1`(2,"OUR")++
1`(3,"ED ")++
1`(4,"PET")++
1`(5,"RI ")++
1`(6,"NET")

Six different tokens

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 colour which is present on the place PacketsToSend

Enabling of SendPacket

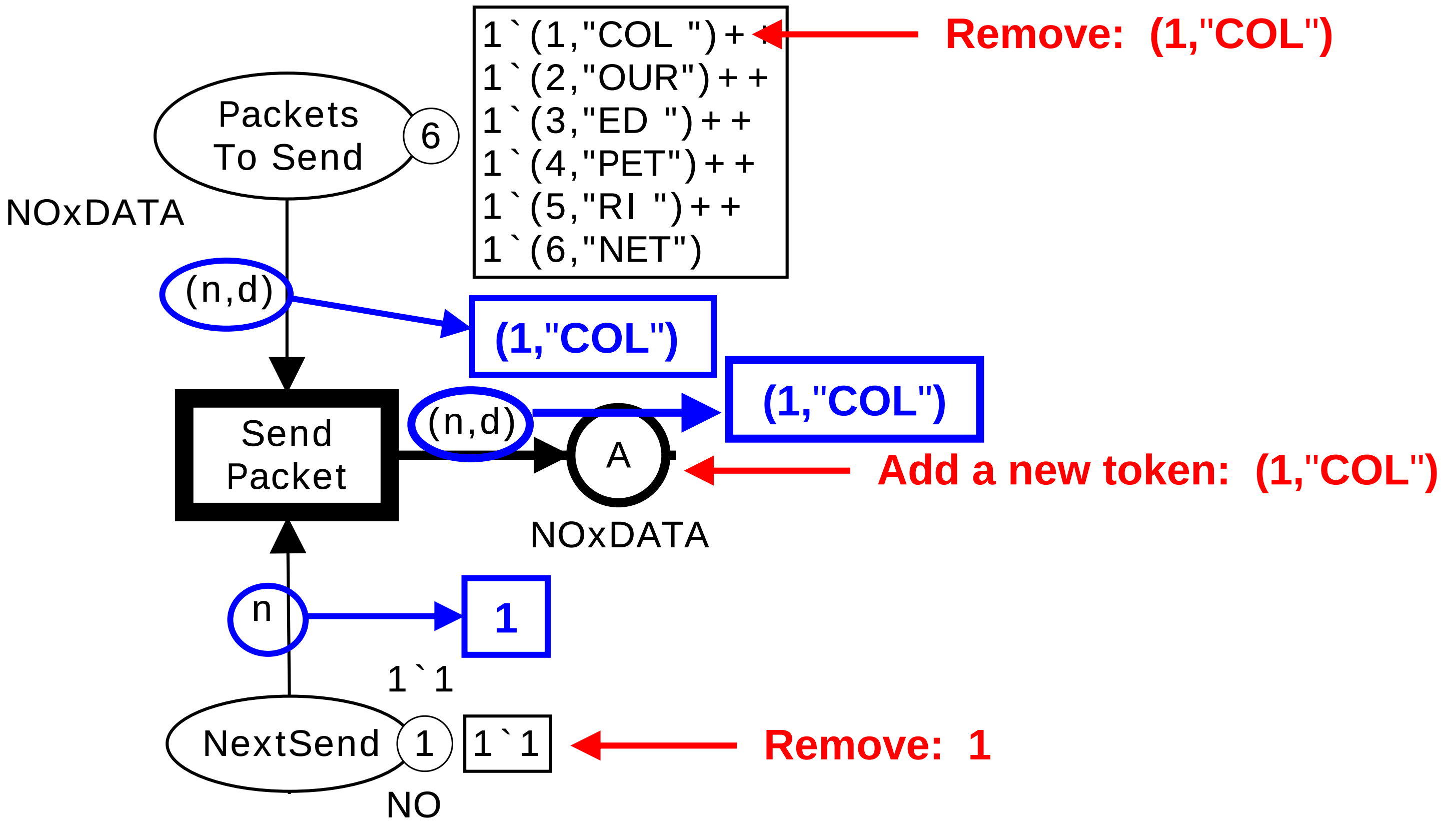


We have found a binding so that each input arc expression evaluates to a colour that is present on the corresponding input place

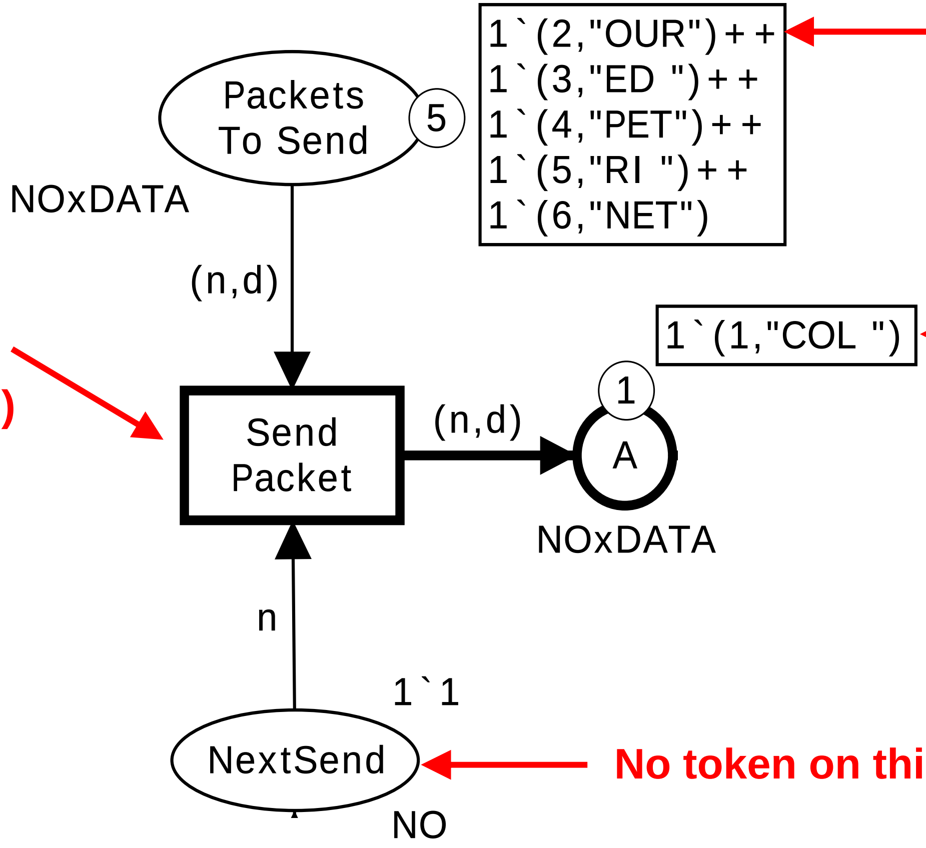
Binding: < n=1 , d="COL" >

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”>



1`(2,"OUR")++
1`(3,"ED ")++
1`(4,"PET")++
1`(5,"RI ")++
1`(6,"NET")

1`(1,"COL ")

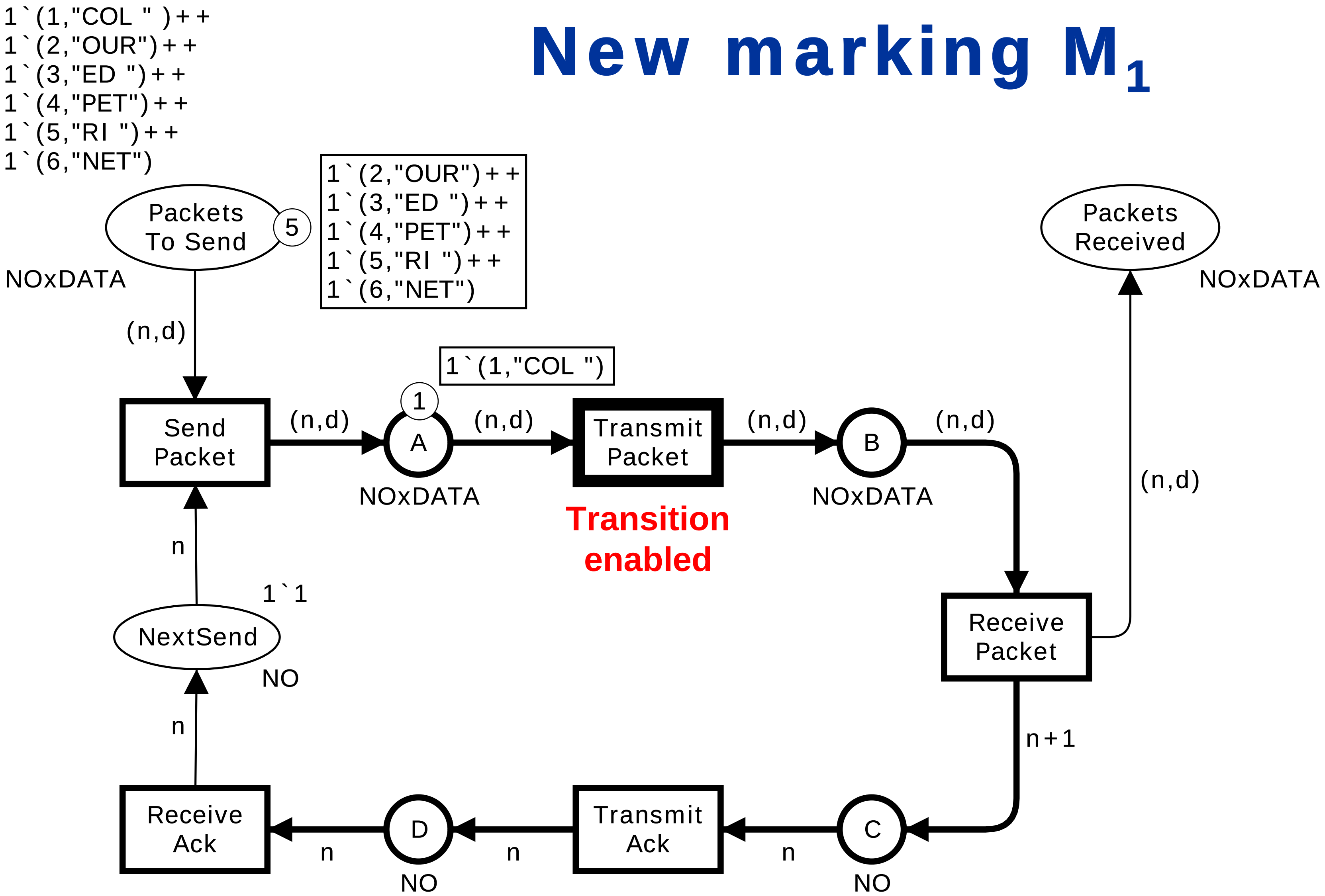
The first packet has been removed

Transition is no longer enabled (thin border line)

A copy of the first packet has been put on A

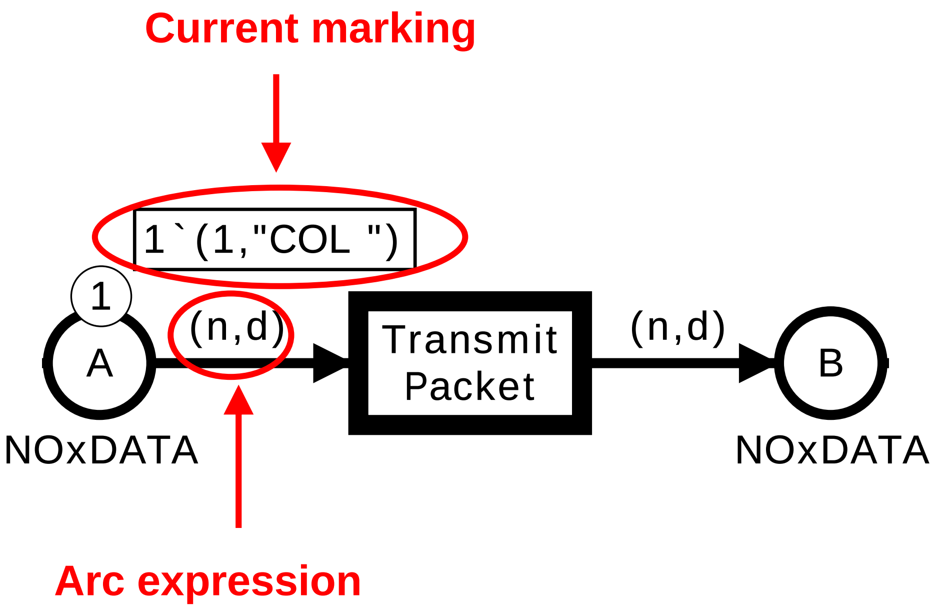
No token on this place

New marking M_1



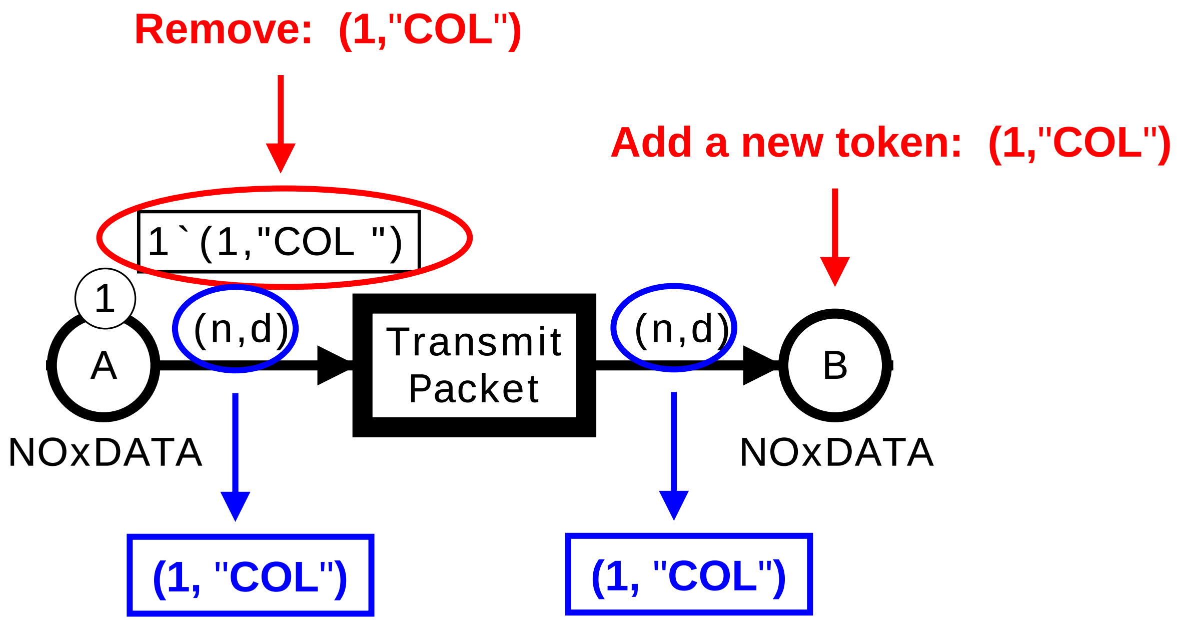
Transition enabled

Binding of TransmitPacket

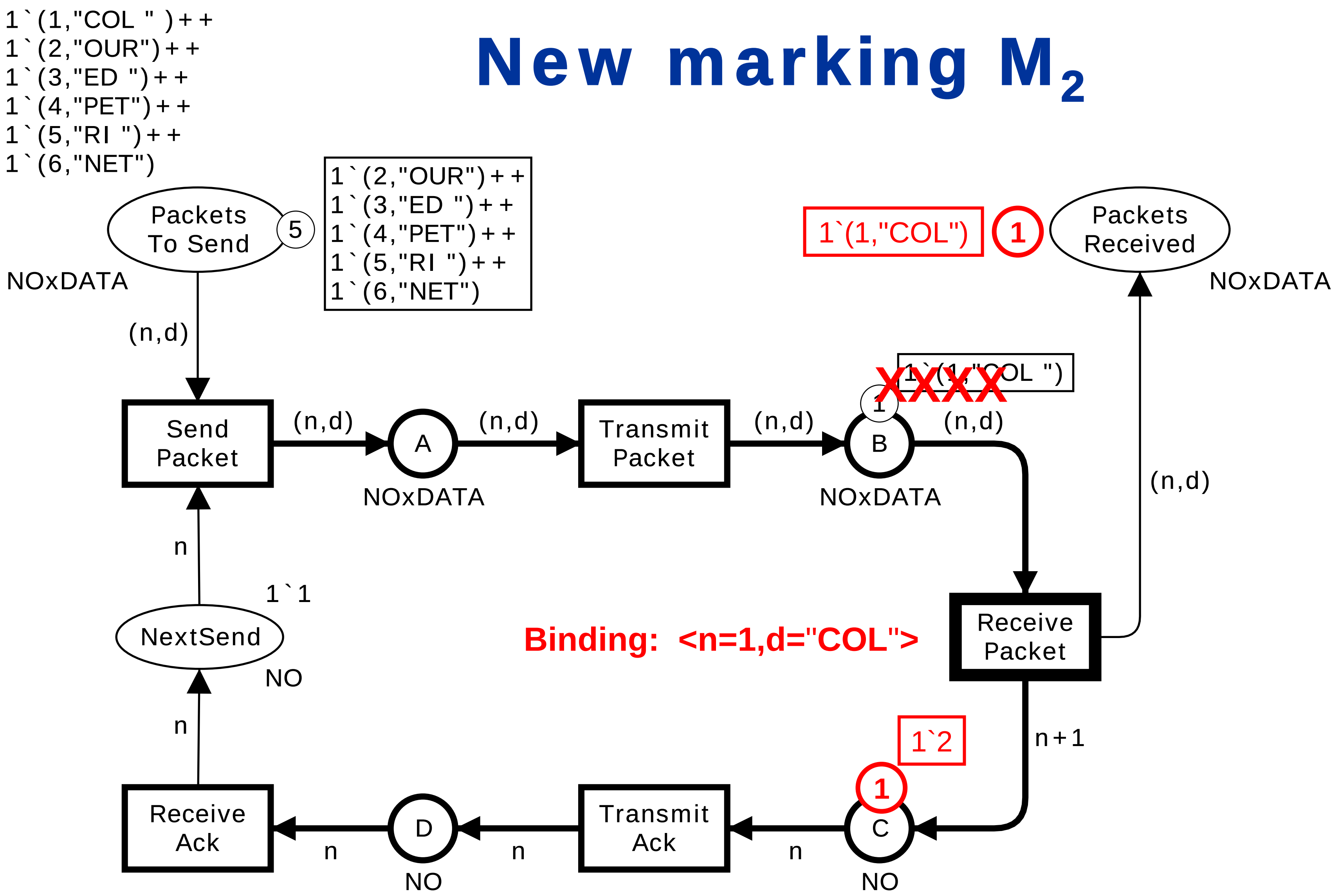


Binding: < n=1 , d="COL" >

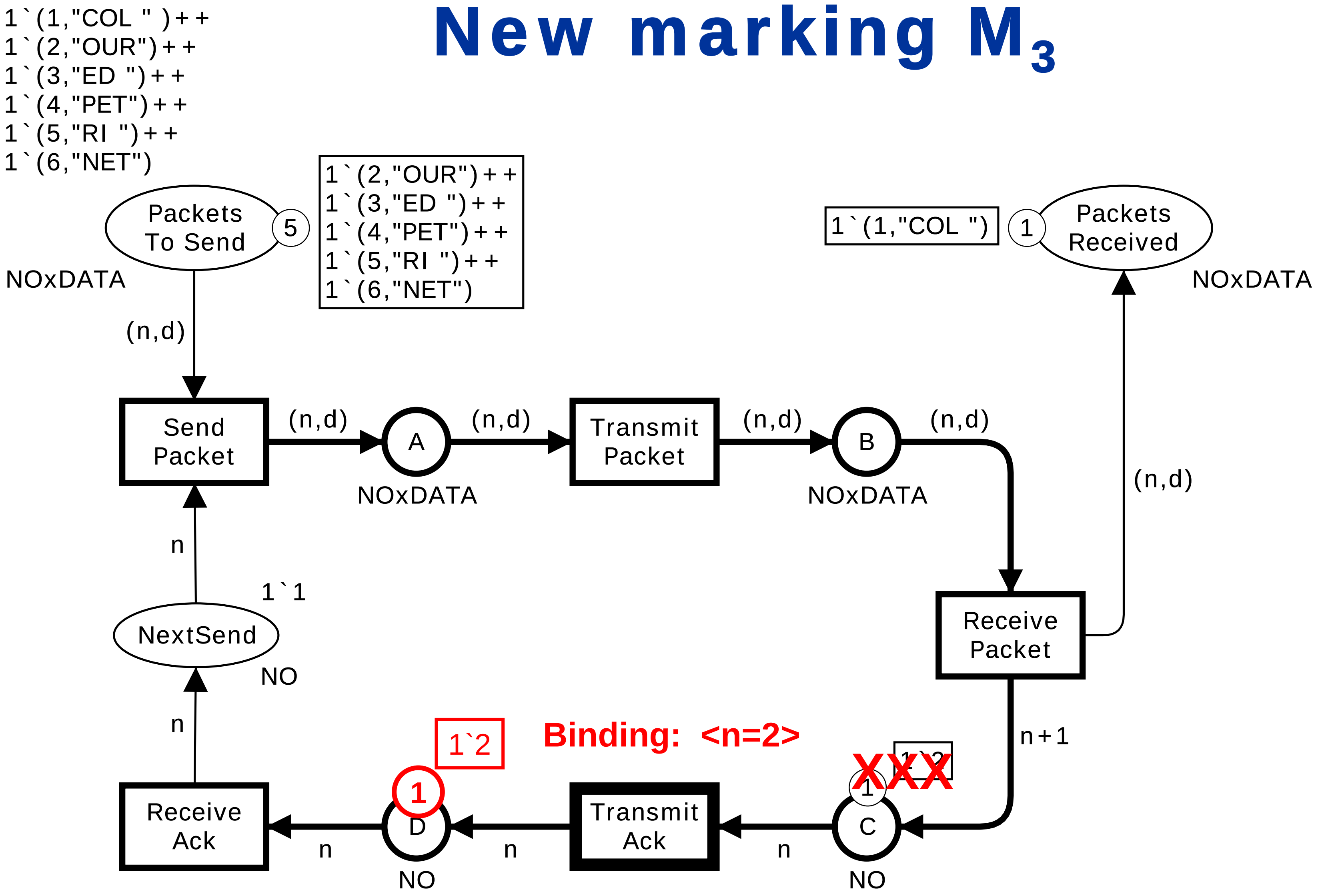
Occurrence of TransmitPacket in binding <n=1,d=“COL”>



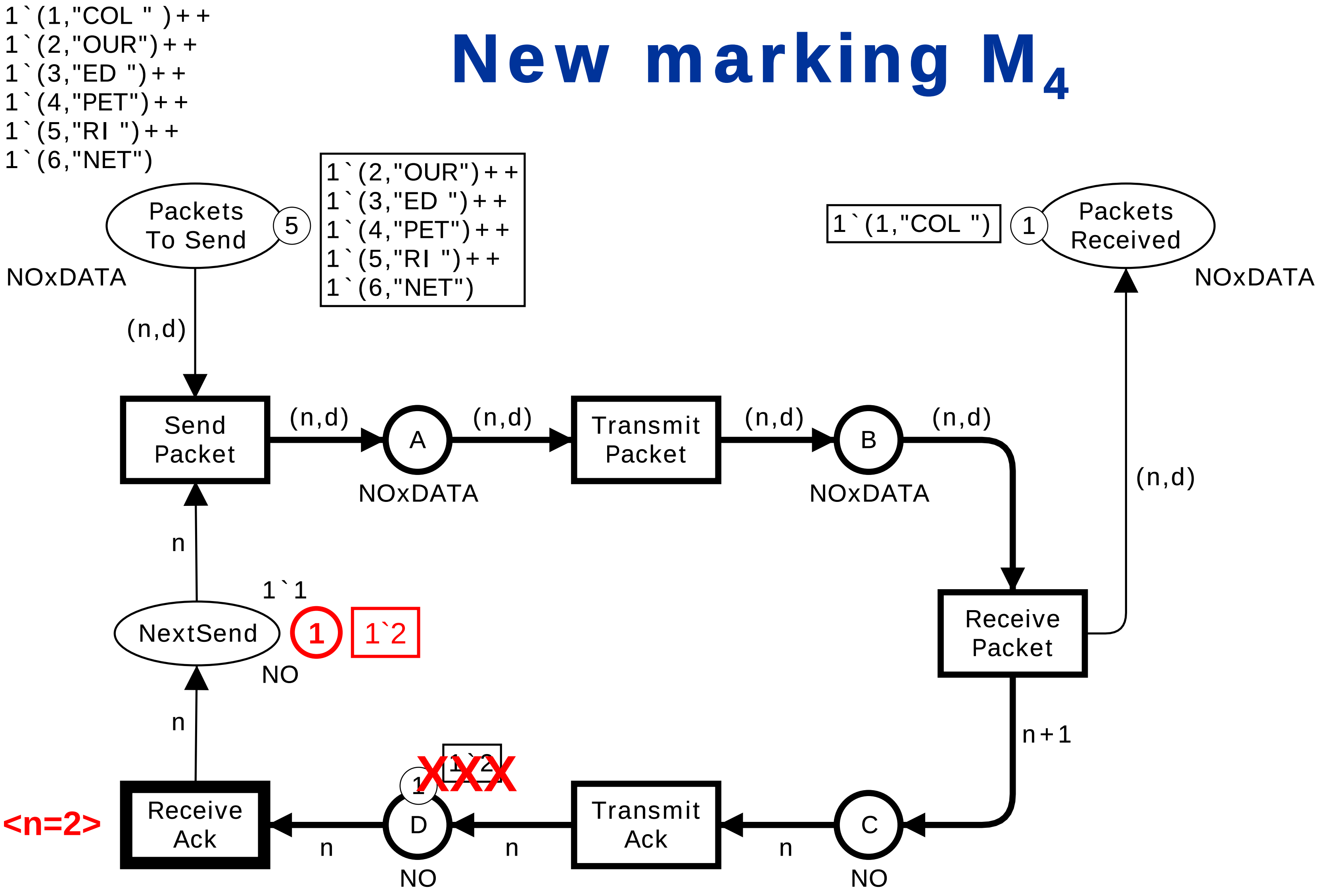
New marking M_2



New marking M_3

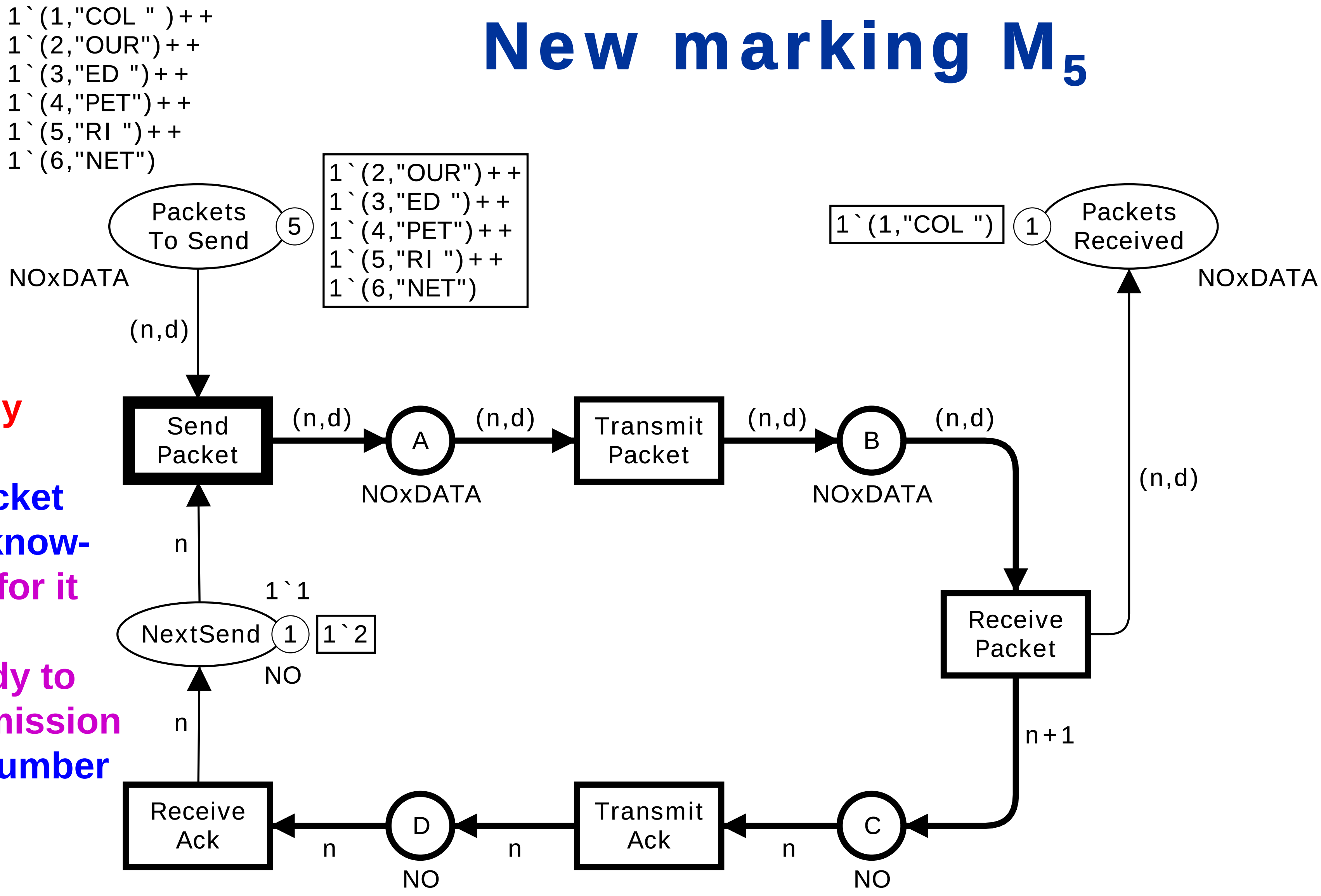


New marking M_4



Binding: <n=2>

New marking M_5



We have **successfully** transmitted **the first packet** and the **acknowledgement** for it

We are ready to **start transmission** of **packet number two**

First five steps

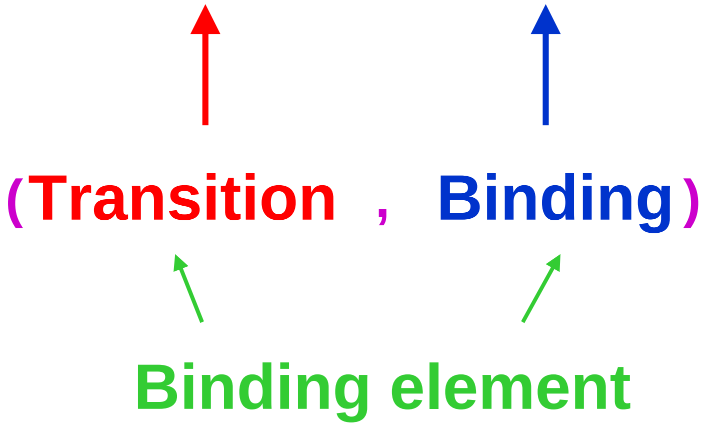
1 (SendPacket,<n=1, d="COL">)

2 (TransmitPacket, <n=1, d="COL">)

3 (ReceivePacket, <n=1, d="COL">)

4 (TransmitAck, <n=2>)

5 (ReceiveAck, <n=2>)

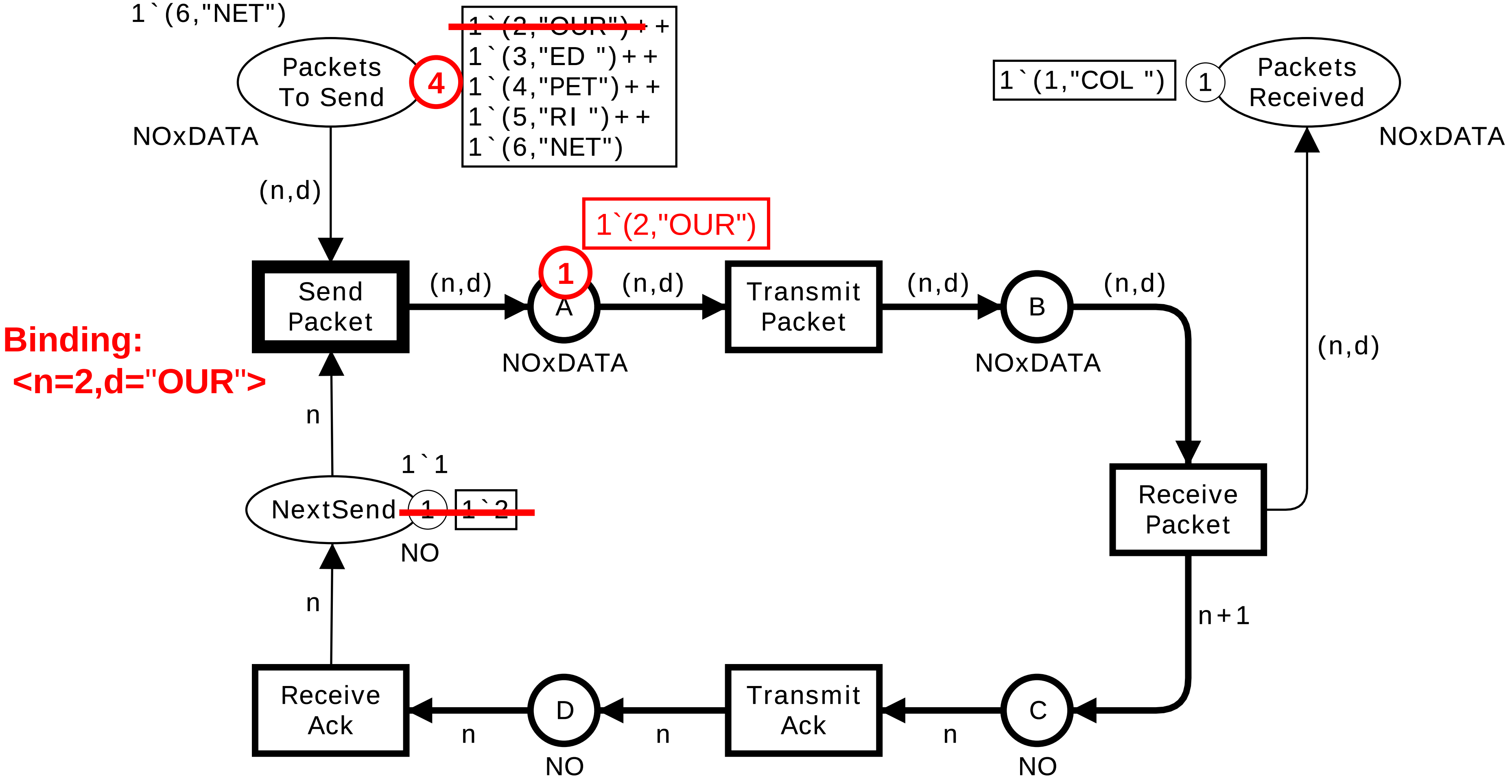


(Transition , Binding)

Binding element

Marking M_5

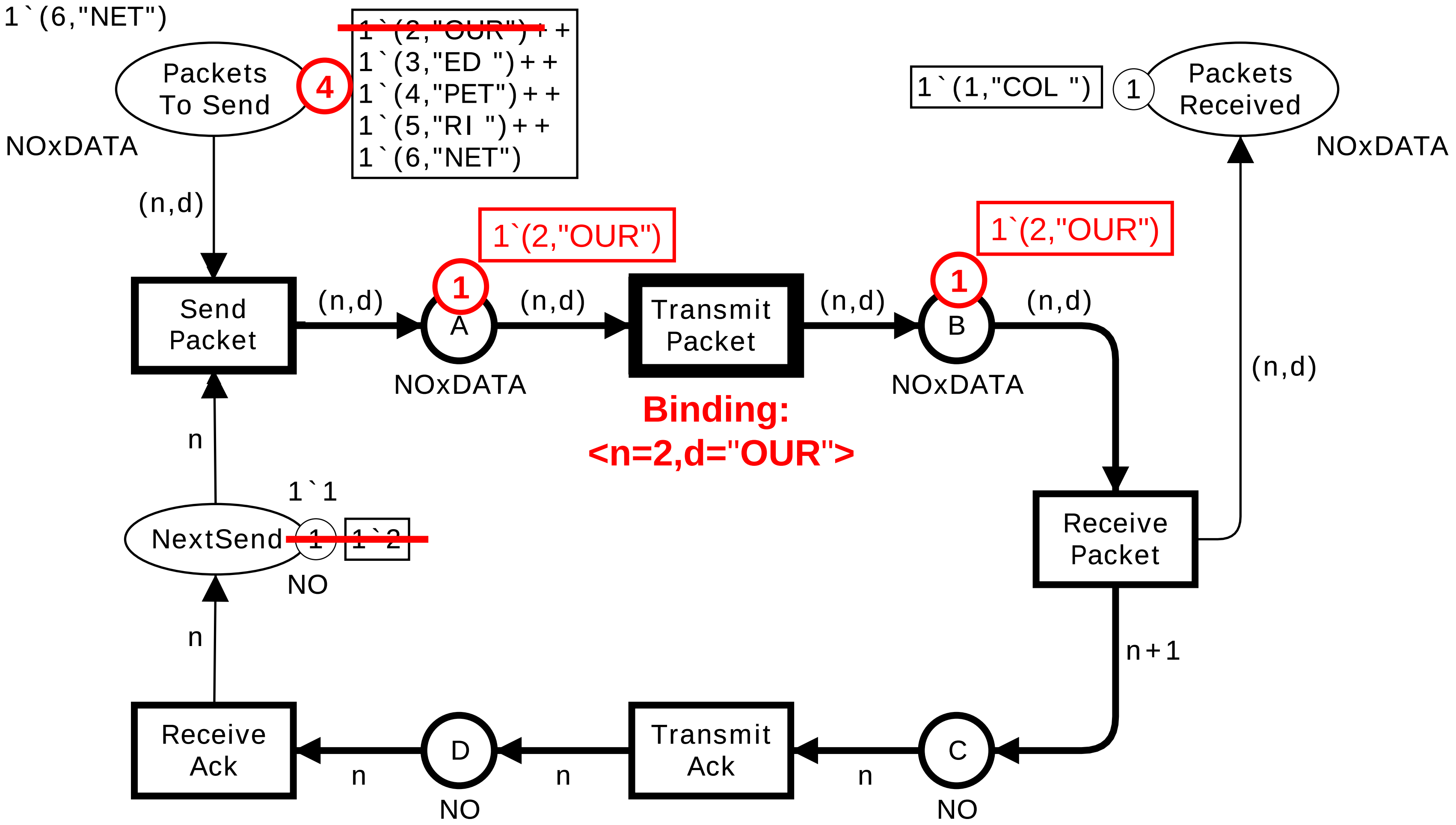
1`(1,"COL ")++
1`(2,"OUR")++
1`(3,"ED ")++
1`(4,"PET")++
1`(5,"RI ")++
1`(6,"NET")



Binding:
<n=2,d="OUR">

New marking M_6

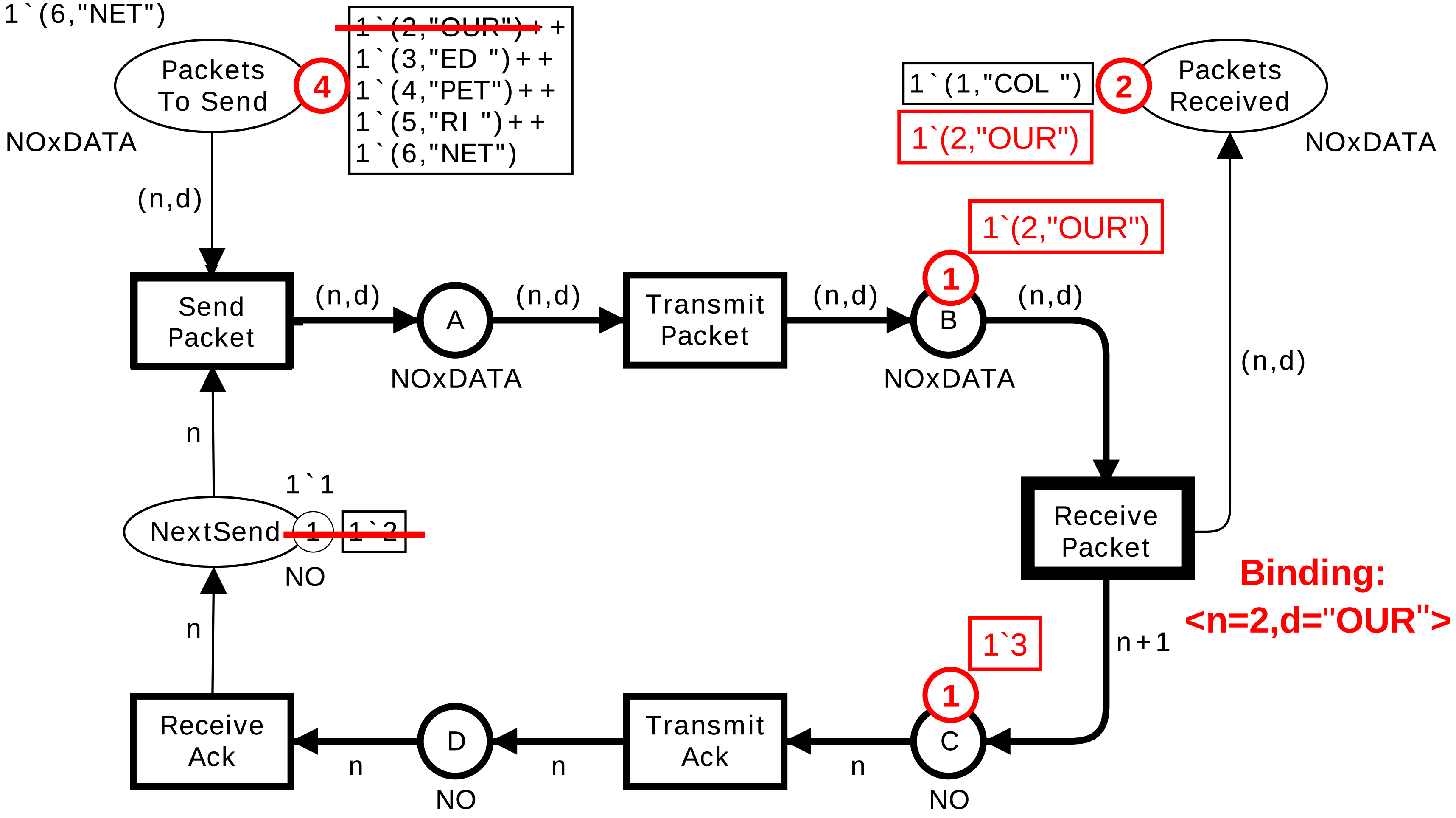
1`(1,"COL ")++
1`(2,"OUR")++
1`(3,"ED ")++
1`(4,"PET")++
1`(5,"RI ")++
1`(6,"NET")



Binding:
<n=2,d="OUR">

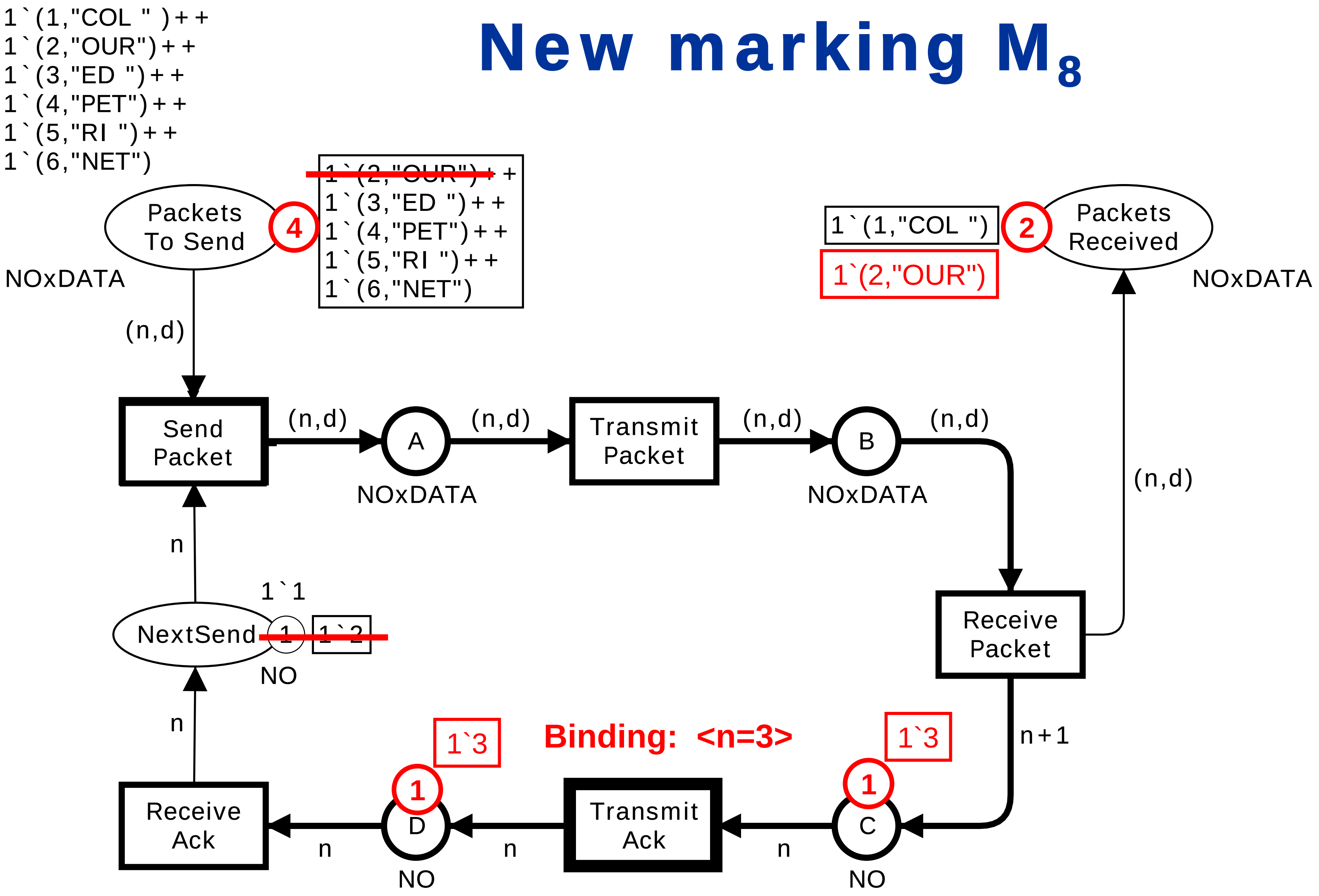
New marking M_7

1`(1,"COL ")++
1`(2,"OUR")++
1`(3,"ED ")++
1`(4,"PET")++
1`(5,"RI ")++
1`(6,"NET")



Binding:
<n=2,d="OUR">

New marking M_8



1`(1,"COL ")++
1`(2,"OUR")++
1`(3,"ED ")++
1`(4,"PET")++
1`(5,"RI ")++
1`(6,"NET")

Packets To Send

4

~~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

(n,d)

B

NOxDATA

(n,d)

Receive Packet

(n,d)

Packets Received

2

1`(1,"COL ")

1`(2,"OUR")

NOxDATA

n+1

n

1`1

NextSend ~~1 1`2~~

NO

n

Receive Ack

n

D

1

1`3

NO

n

Binding: <n=3>

Transmit Ack

n

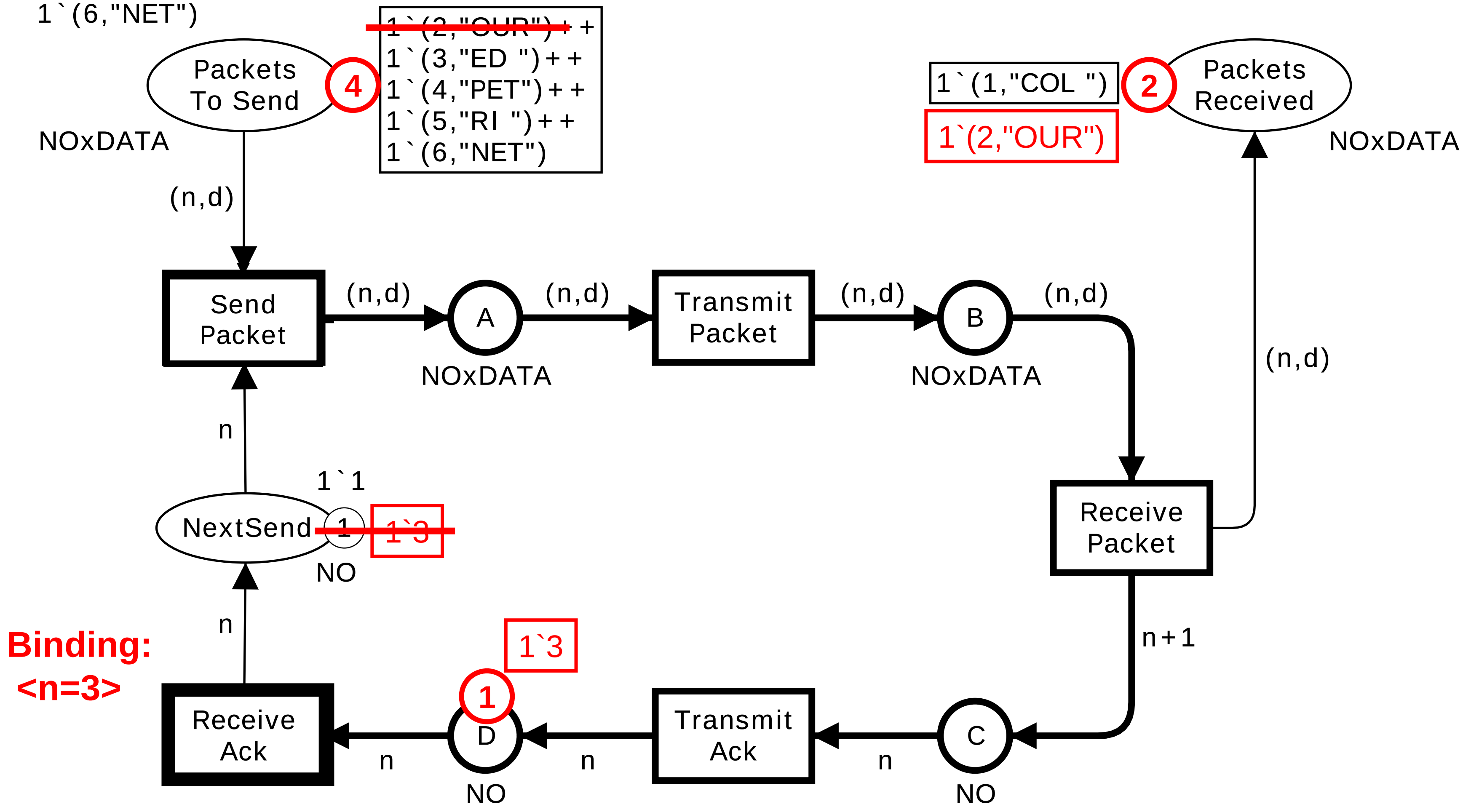
C

1

1`3

NO

New marking M_9



1`(1,"COL ")++
1`(2,"OUR")++
1`(3,"ED ")++
1`(4,"PET")++
1`(5,"RI ")++
1`(6,"NET")

~~1`(2,"OUR")++~~
1`(3,"ED ")++
1`(4,"PET")++
1`(5,"RI ")++
1`(6,"NET")

4

Packets To Send

NOxDATA

(n,d)

Send Packet

(n,d)

A

NOxDATA

(n,d)

Transmit Packet

(n,d)

B

NOxDATA

(n,d)

1`(1,"COL ")

1`(2,"OUR")

2

Packets Received

NOxDATA

(n,d)

Receive Packet

n+1

C

NO

n

Transmit Ack

n

1`3

1

D

NO

n

Receive Ack

n

NextSend

1`1

~~1~~

~~1`3~~

NO

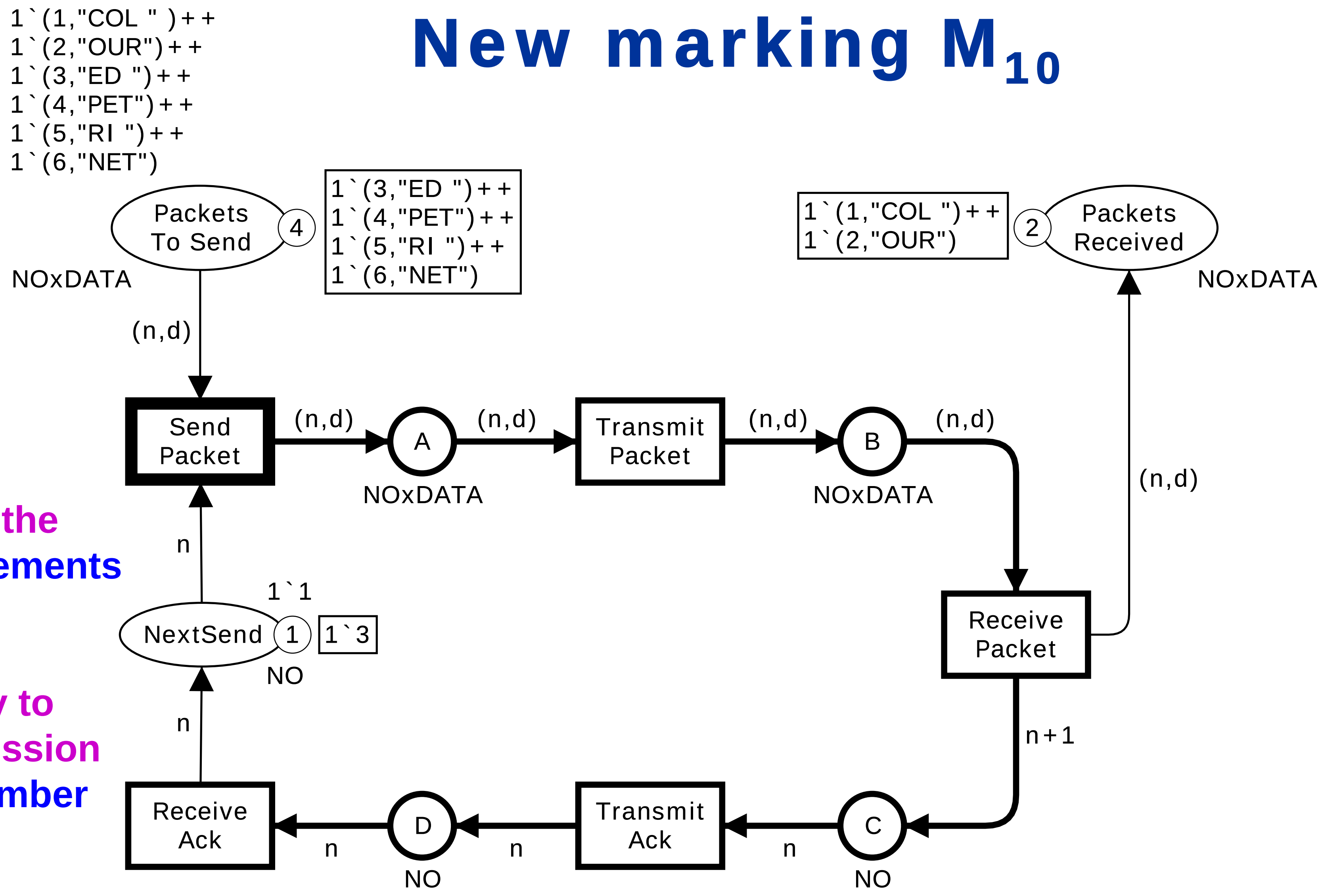
n

Binding: <n=3>

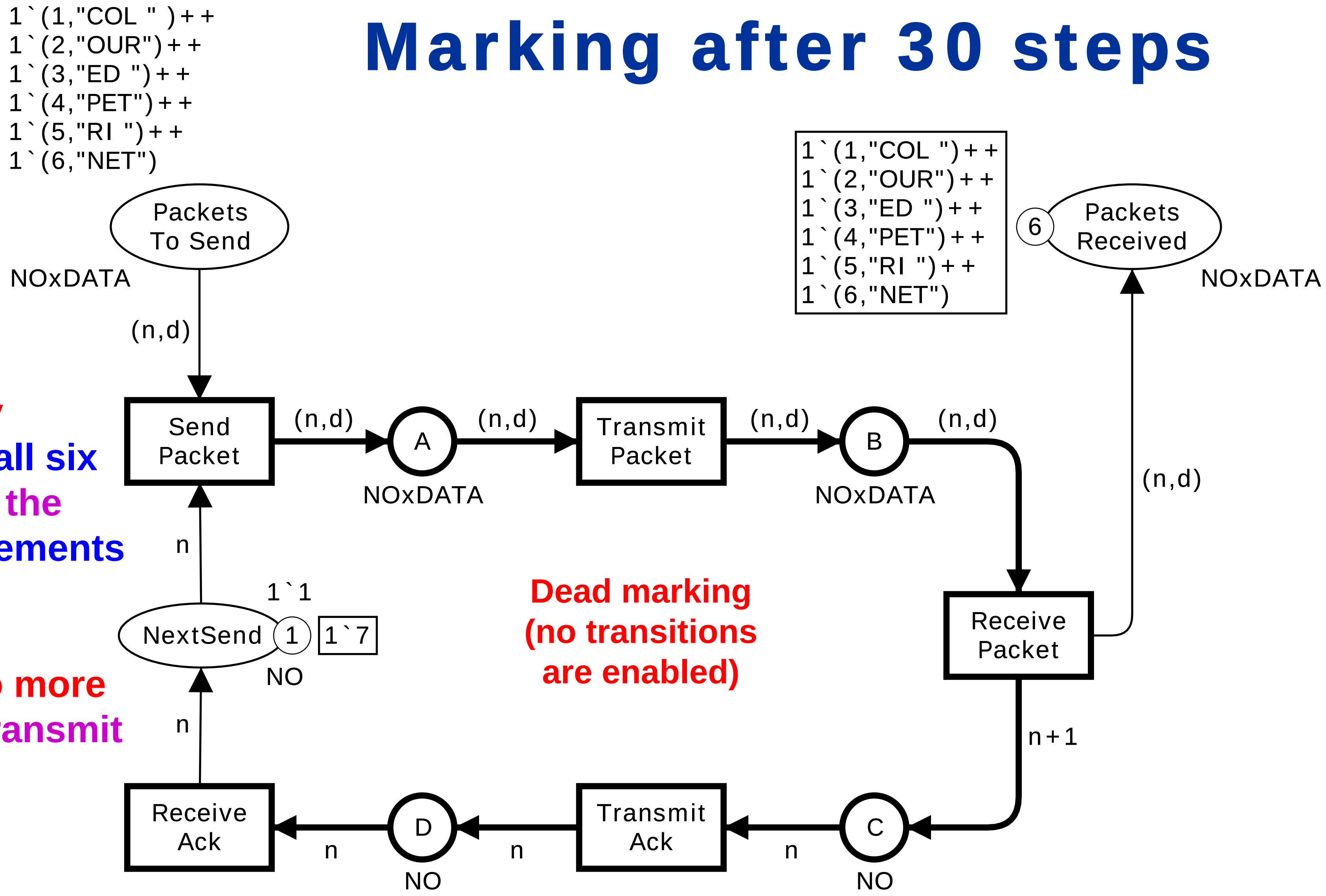
New marking M_{10}

We have successfully transmitted the first two packets and the acknowledgements for them

We are ready to start transmission of packet number three



Marking after 30 steps

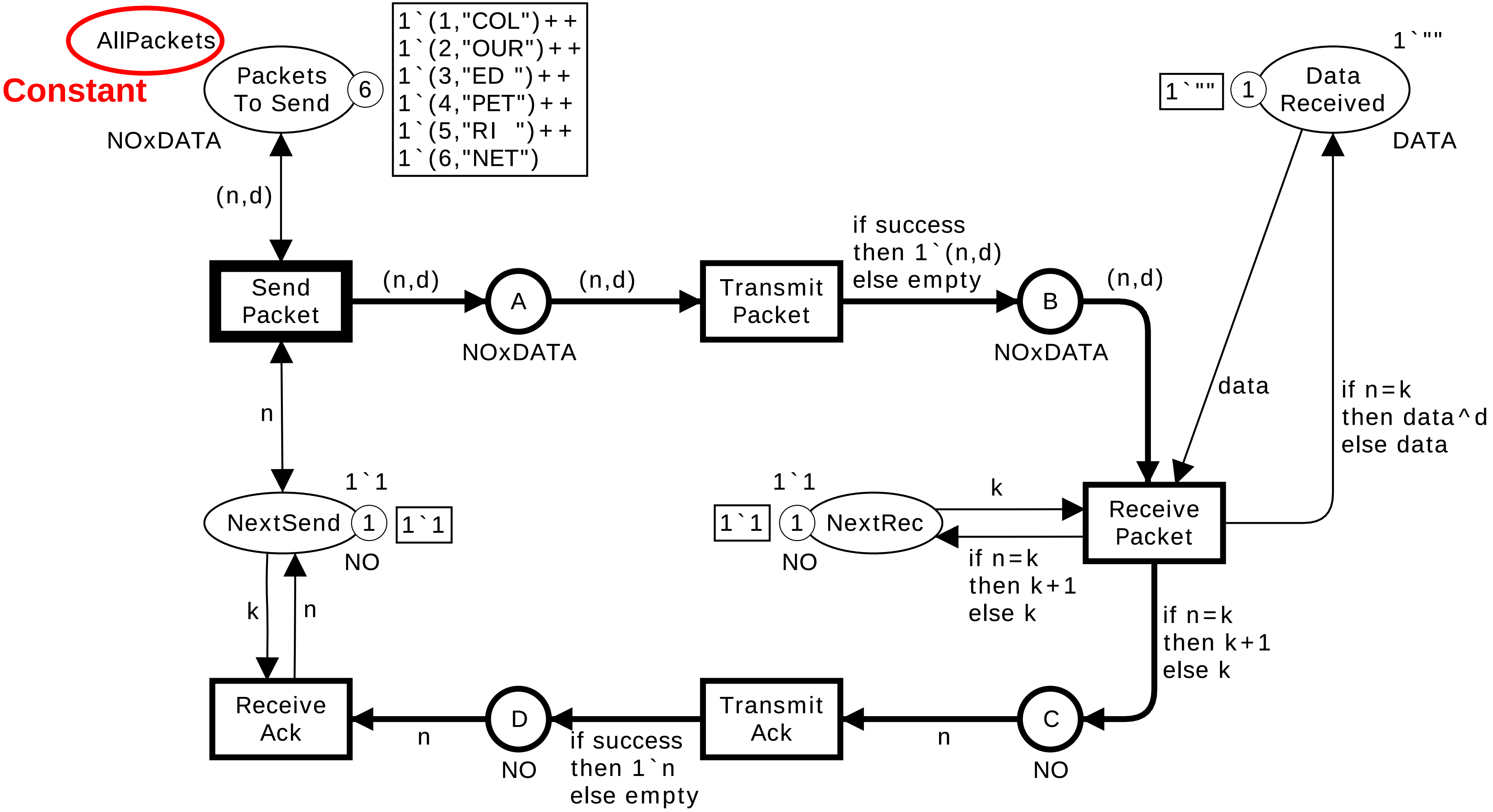


We have successfully transmitted all six packets and the acknowledgements for them

There are no more packets to transmit

Dead marking (no transitions are enabled)

Second version of protocol



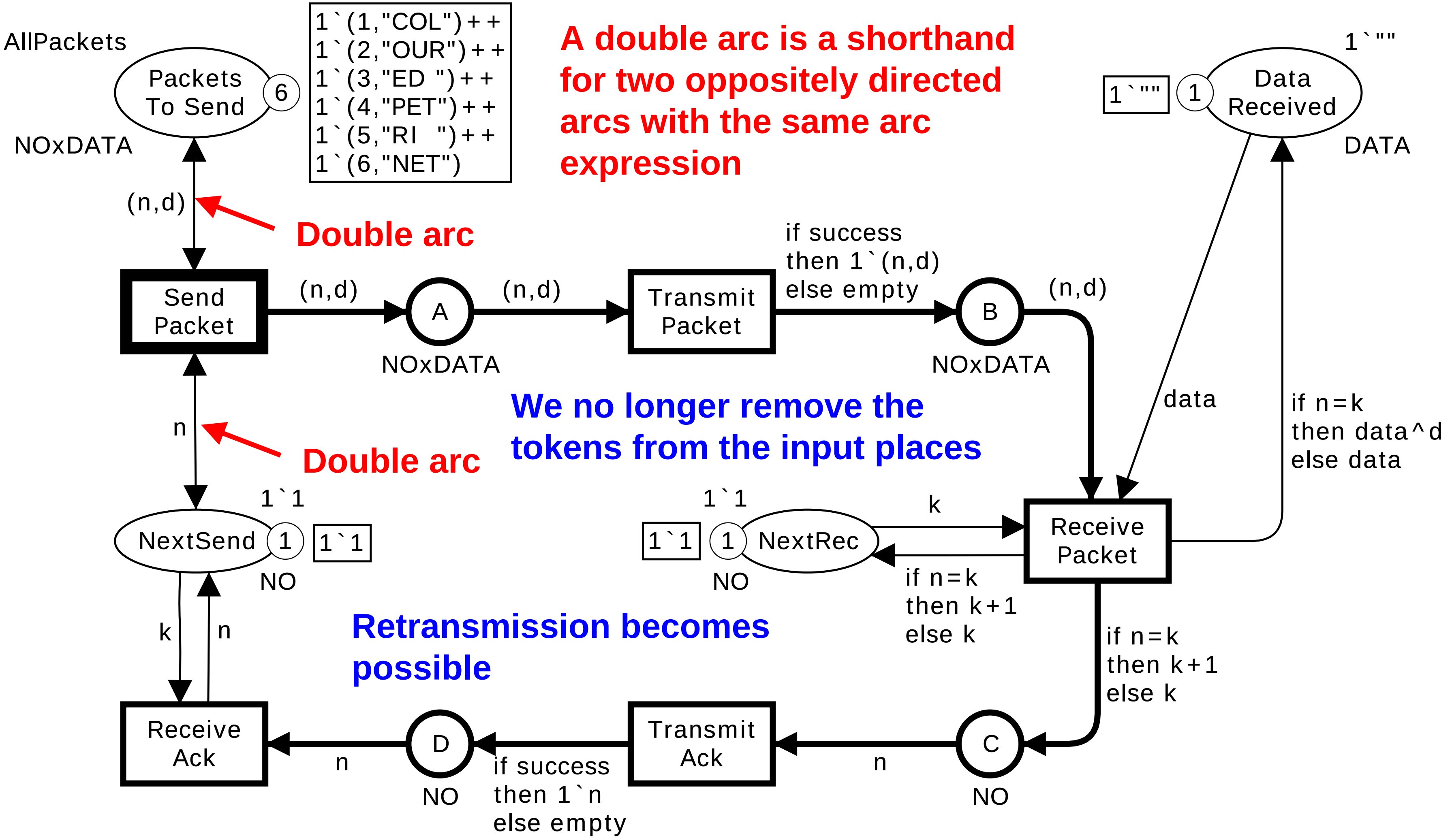
Declaration of constants

- We use the following constant to specify the initial marking of PacketsToSend.

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

- Saves a little bit of space in the diagram.
- Enhances readability.
- Can be reused (at other places).

Double arcs



A double arc is a shorthand for two oppositely directed arcs with the same arc expression

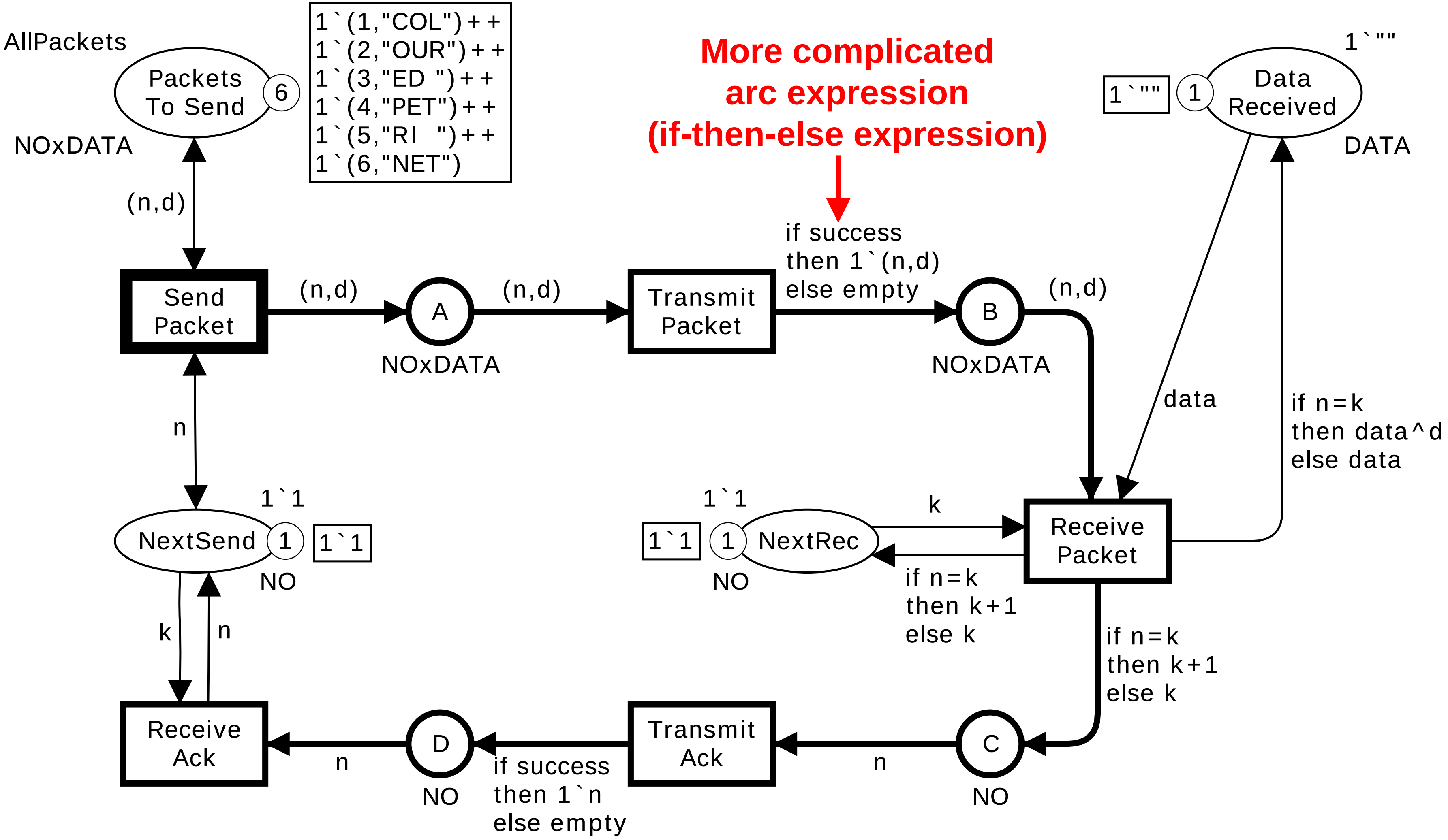
Double arc

Double arc

We no longer remove the tokens from the input places

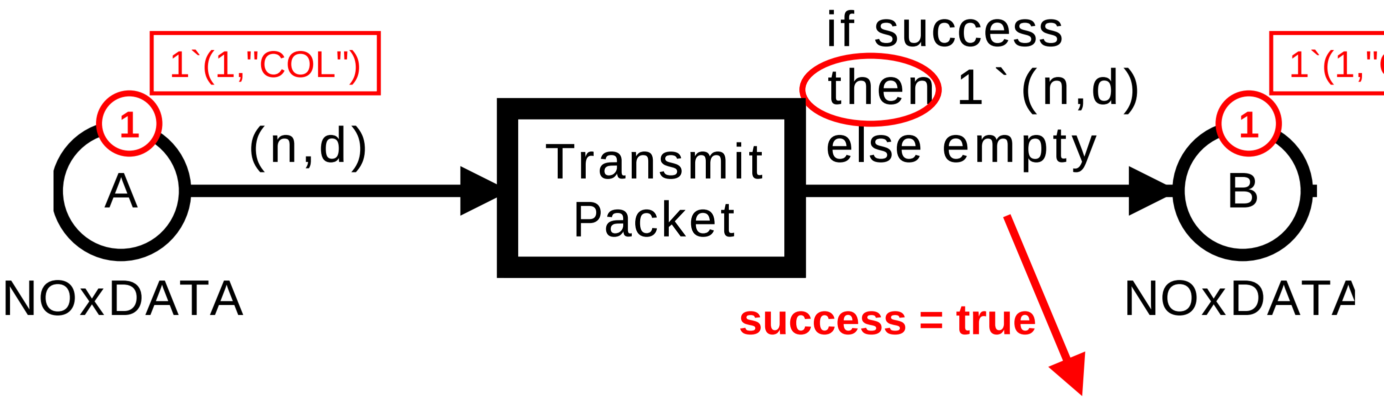
Retransmission becomes possible

More complicated arc expression



More complicated arc expression (if-then-else expression)

If-then-else expression



1`(1,"COL")

New variable:

var success : BOOL;

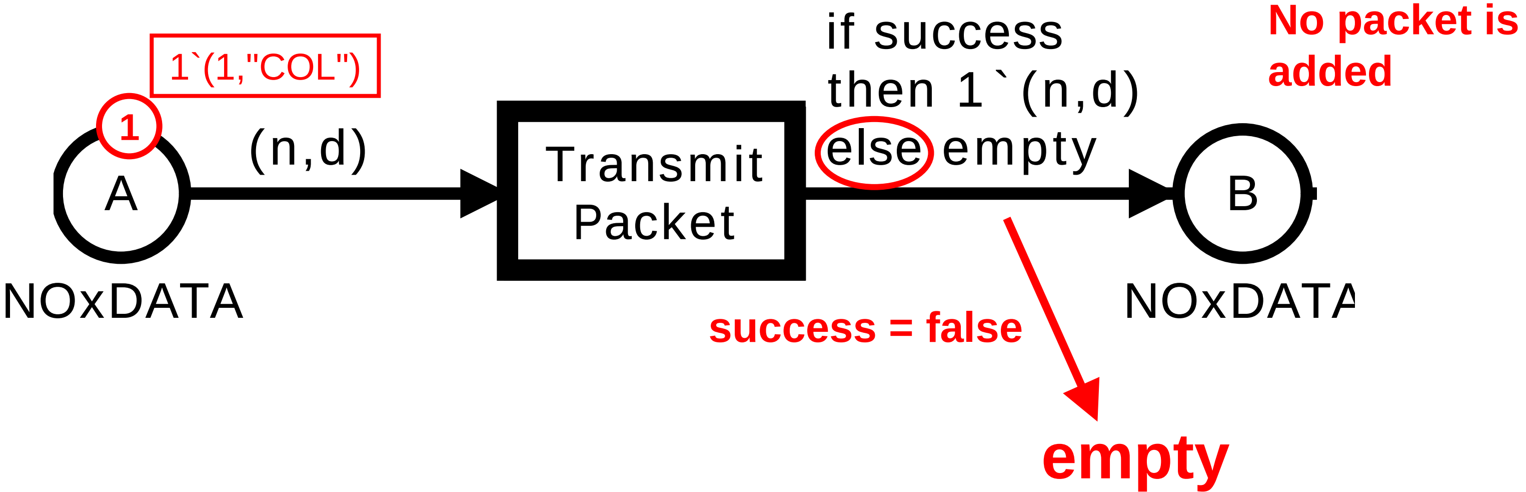
Successful transmission over the network

→

b^+ = <n=1, d="COL", success=true>

b^- = <n=1, d="COL", success=false>

If-then-else expression



No packet is added

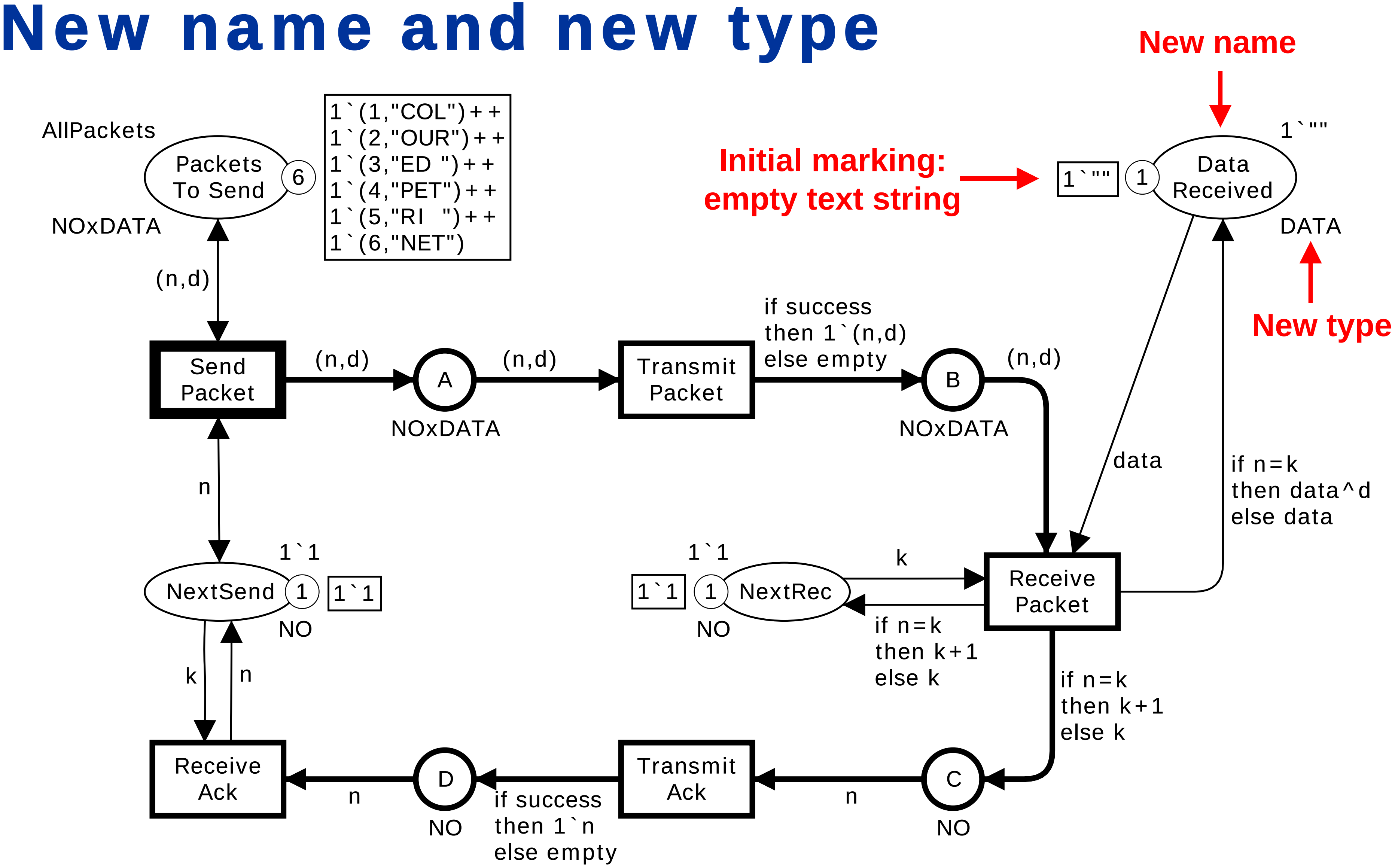
var success : BOOL;

Packet is lost during transmission

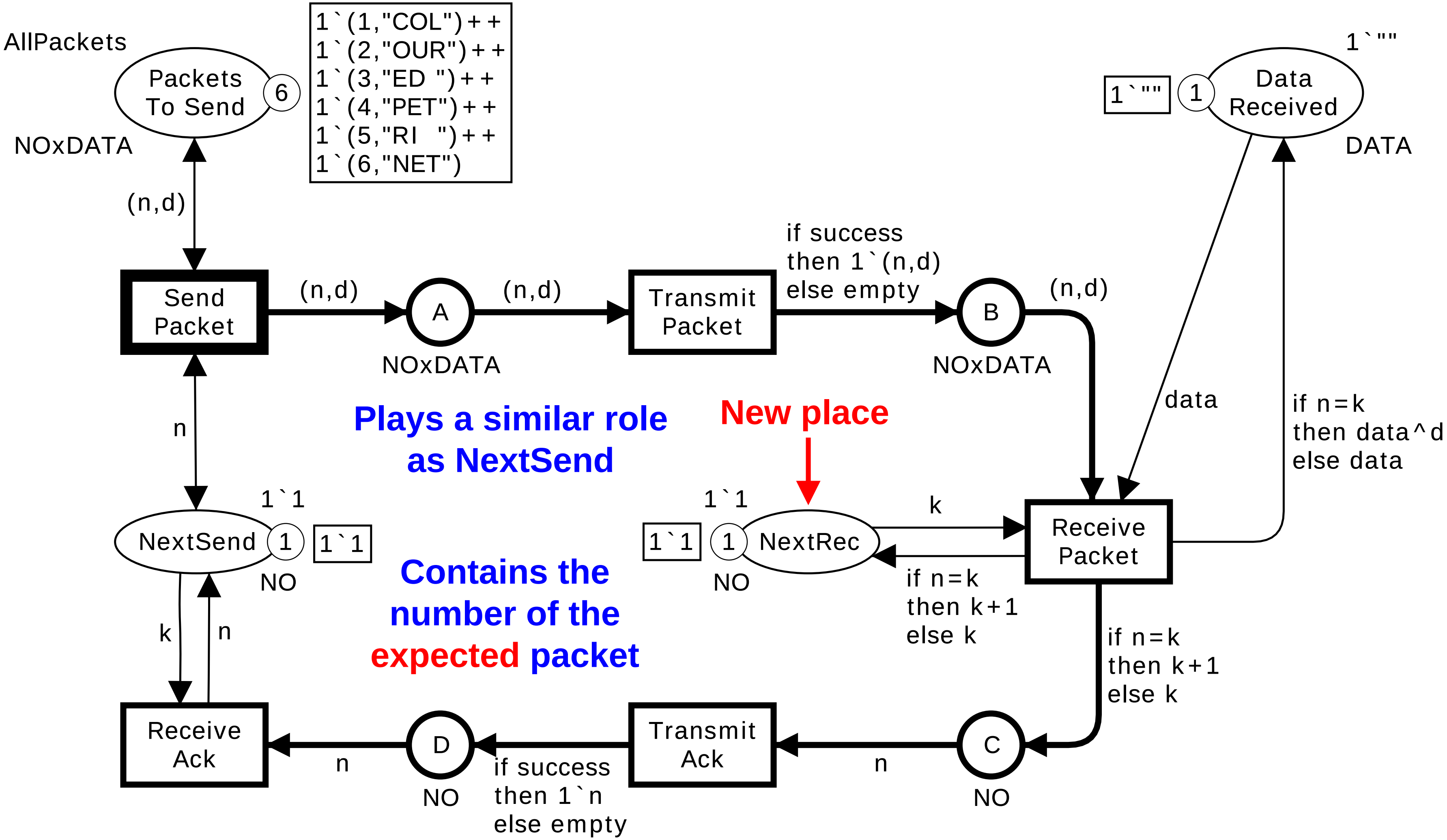
b^{+} = <n=1, d="COL", success=true>

b^{-} = <n=1, d="COL", success=false>

New name and new type



New place: NextRec

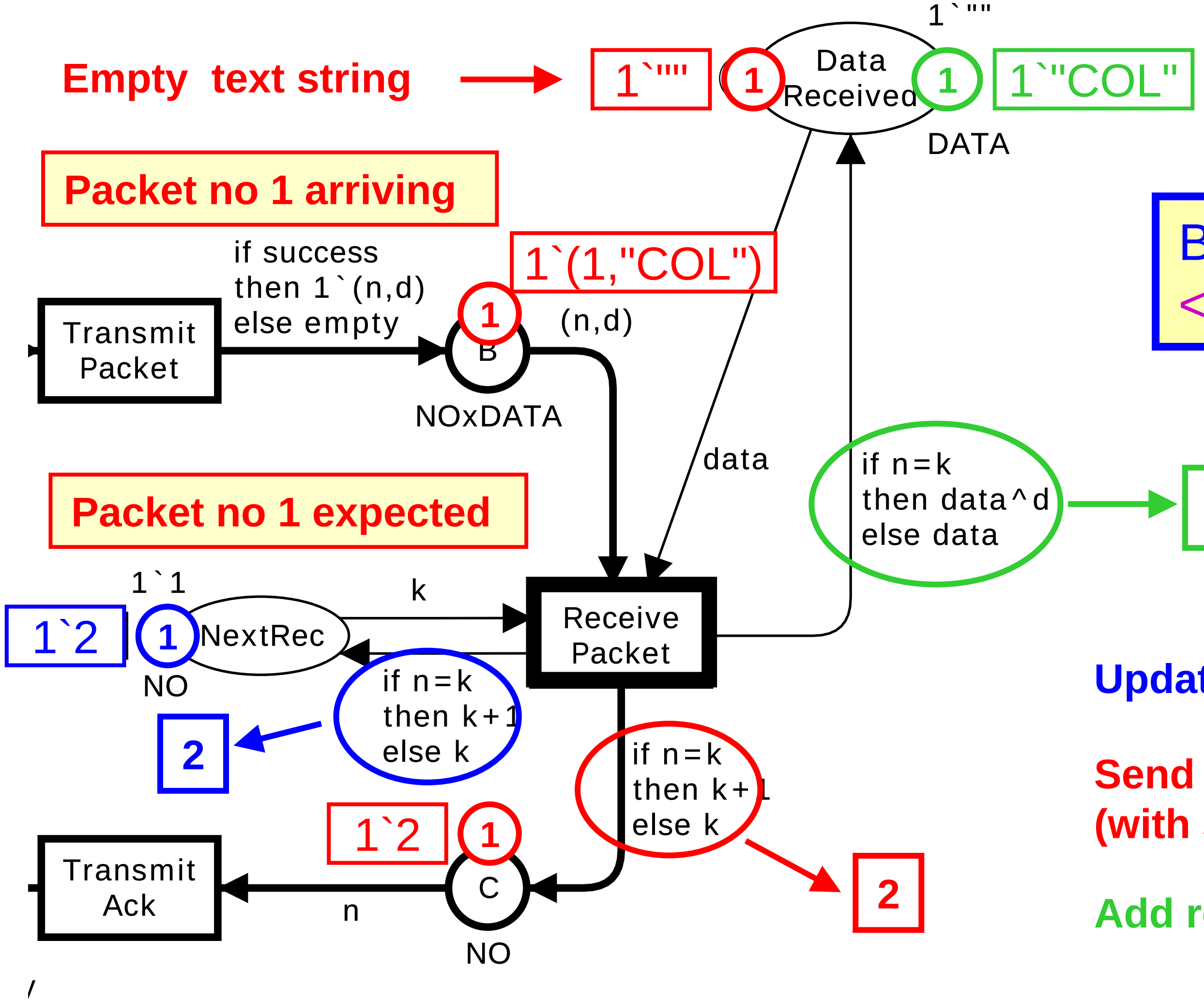


Plays a similar role as NextSend

New place

Contains the number of the expected packet

Correct packet arrives



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

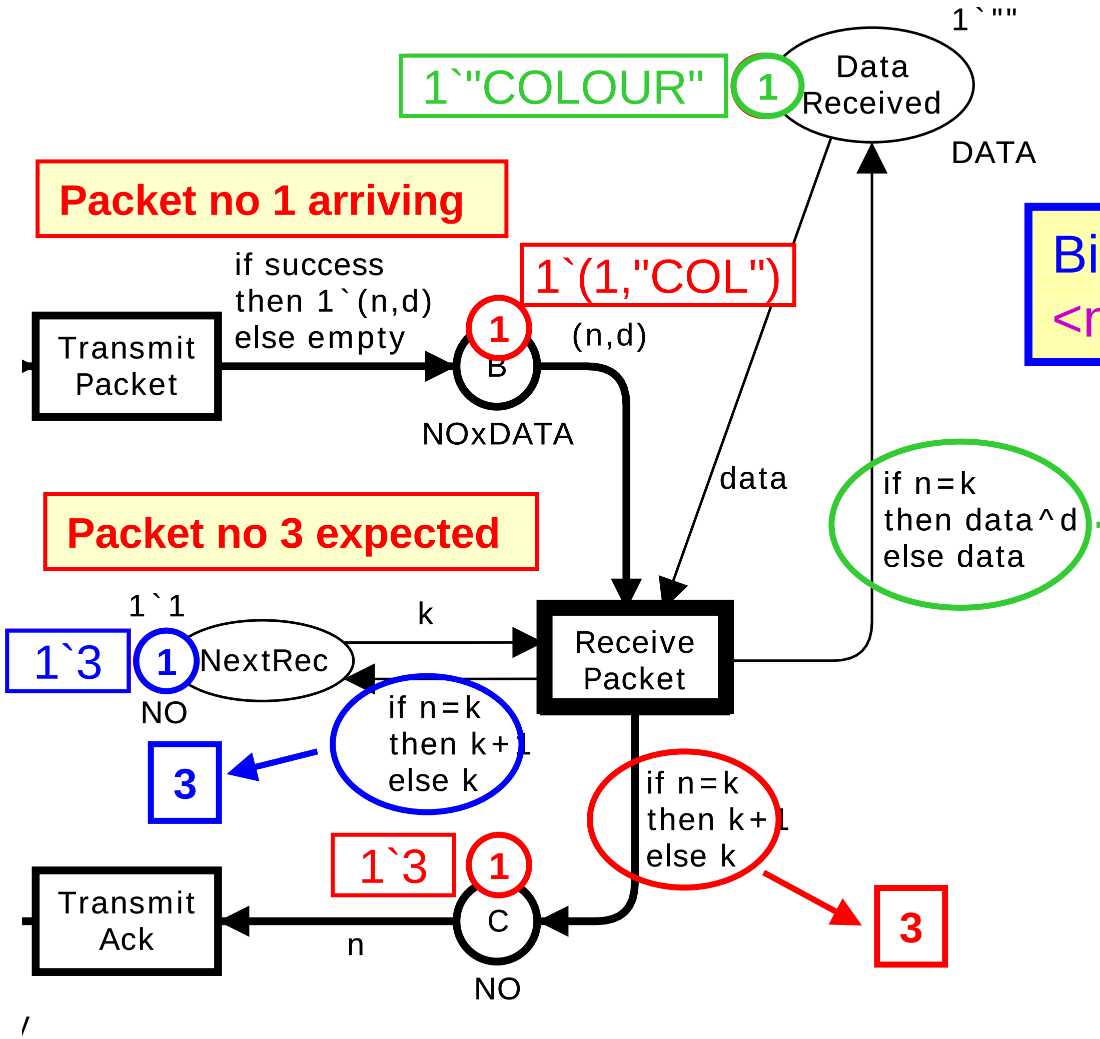
^ is the concatenation operator

Update NextRec (from 1 to 2)

Send acknoweledgement
(with sequence number of next packet)

Add received data: "COL"

Wrong packet arrives



"COLOUR"

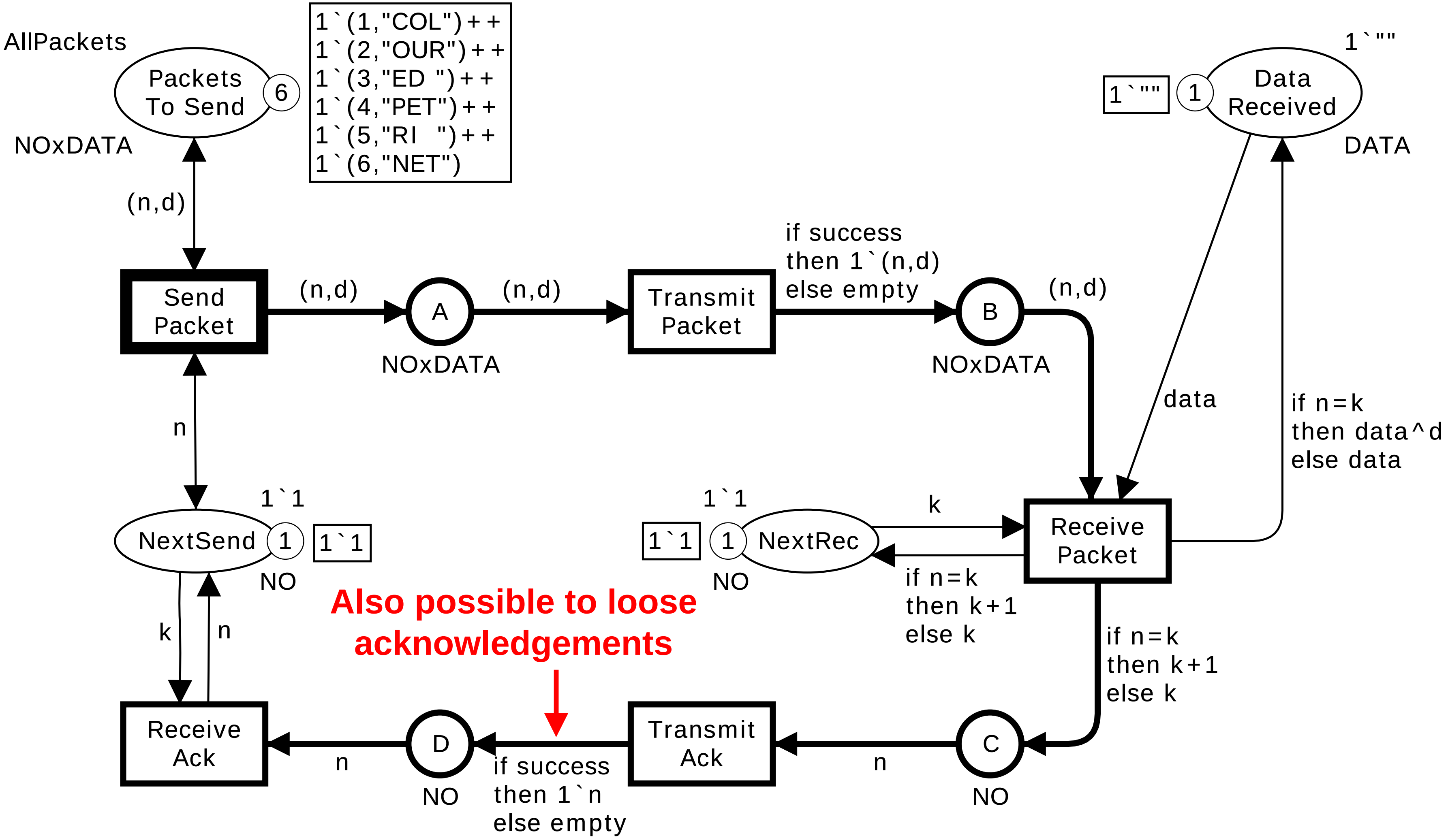
Binding:
<n=1, d="COL", k=3, data="COLOUR">

Do not change NextRec

Send acknowledgement
(with sequence number of next packet)

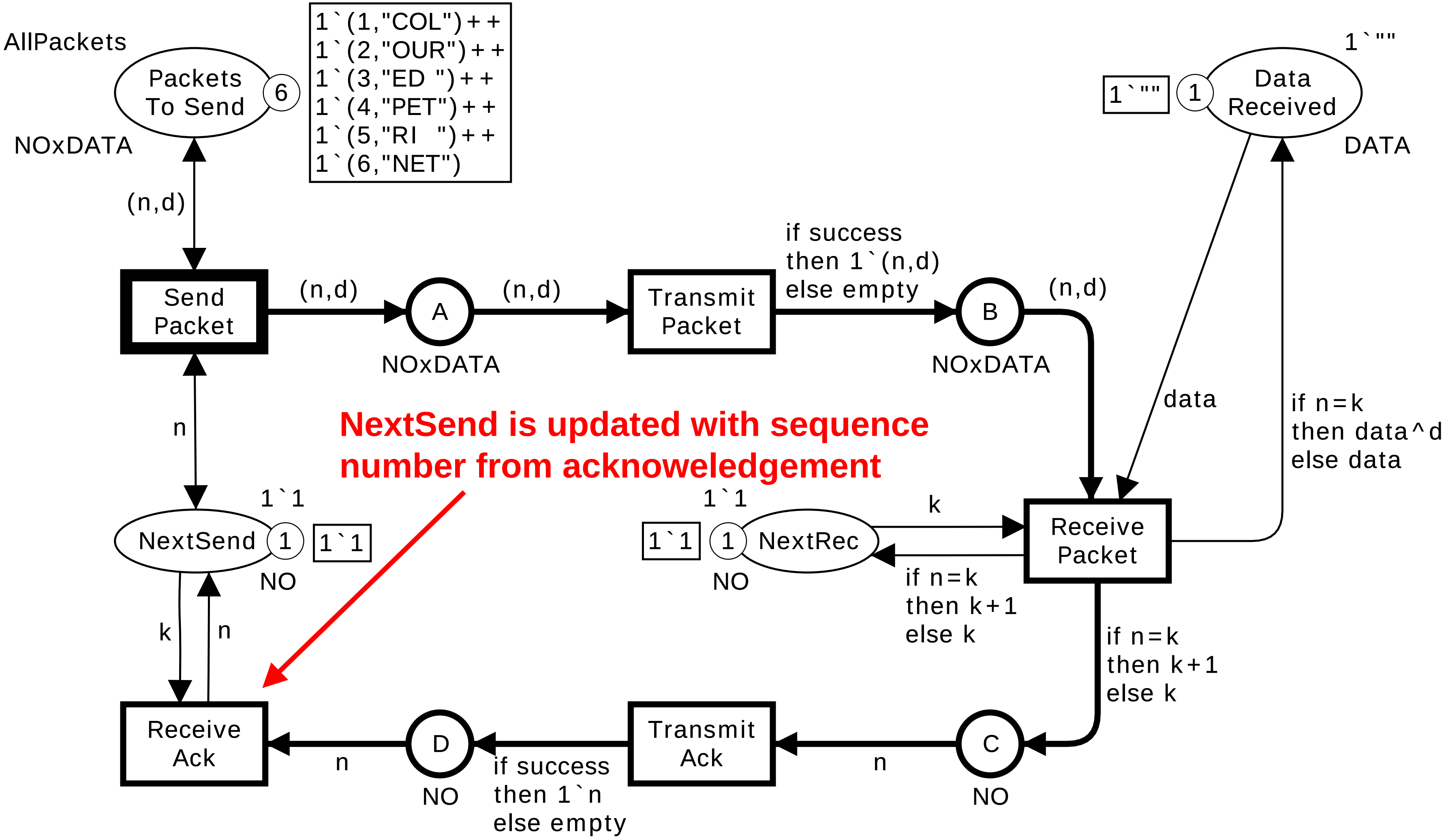
No data is added to DataReceived

Acknowledgements can be lost



Also possible to loose acknowledgements

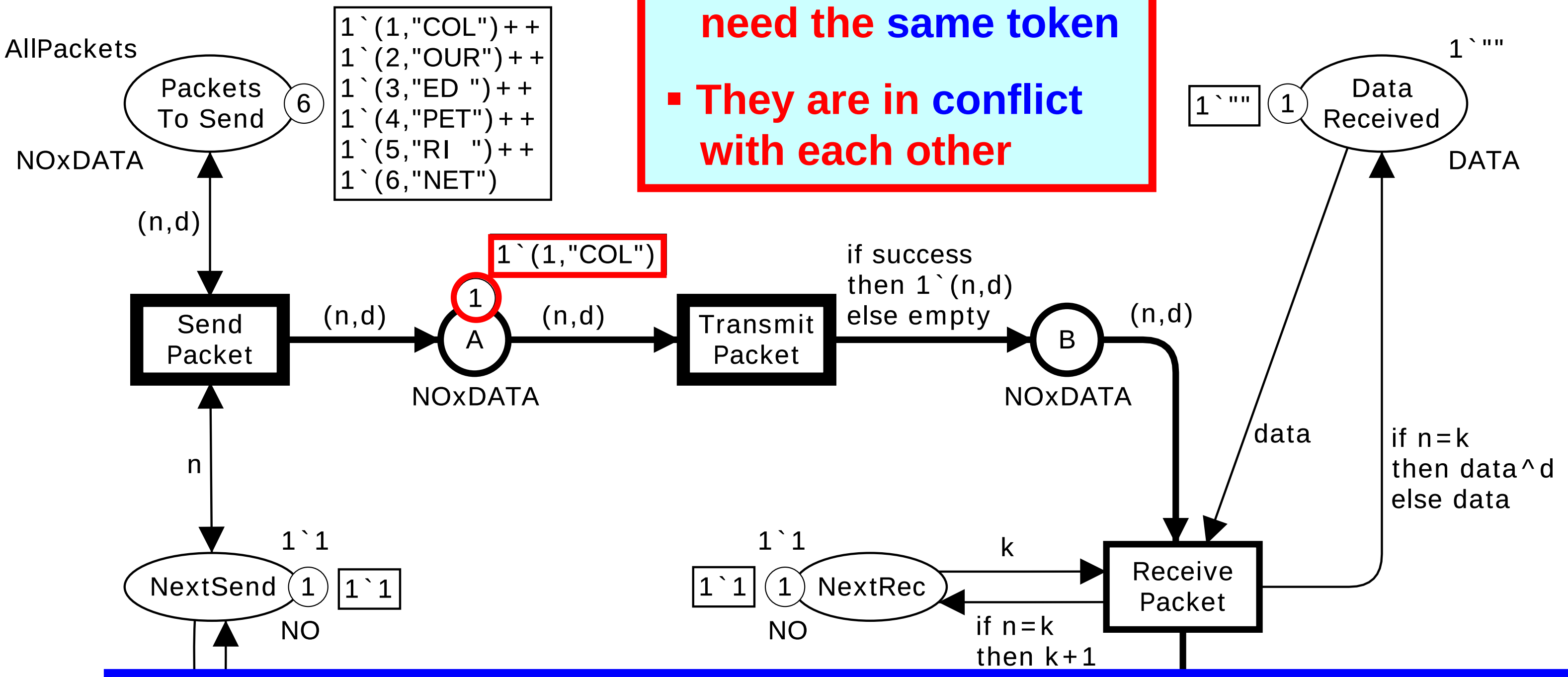
NextSend is updated



NextSend is updated with sequence number from acknoweledgement

Two enabled transitions

- These bindings need the same token
- They are in conflict with each other



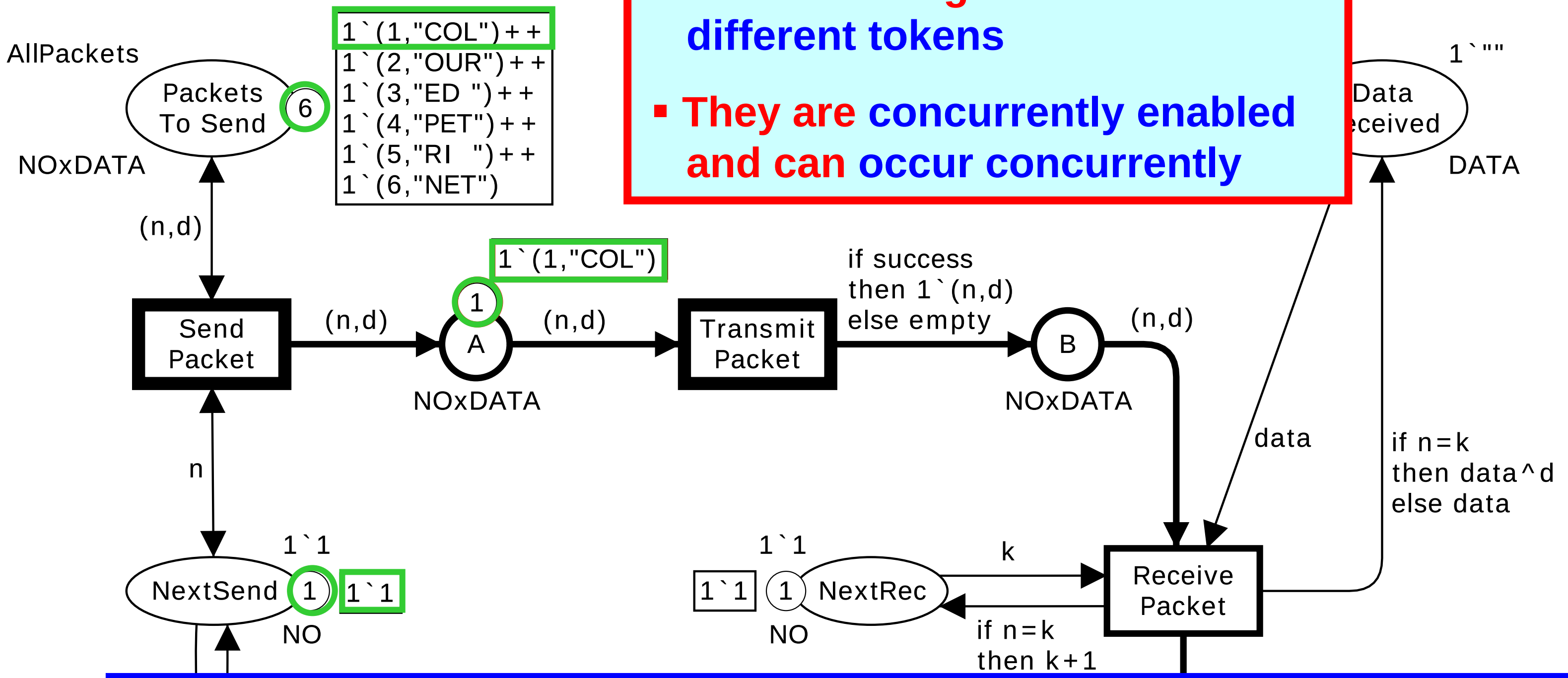
SP = (SendPacket, <n=1, d="COL">)

● TP$^+$ = (TransmitPacket, <n=1, d="COL", success=true>)

● TP$^-$ = (TransmitPacket, <n=1, d="COL", success=false>)

Two enabled transitions

- These bindings use different tokens
- They are concurrently enabled and can occur concurrently

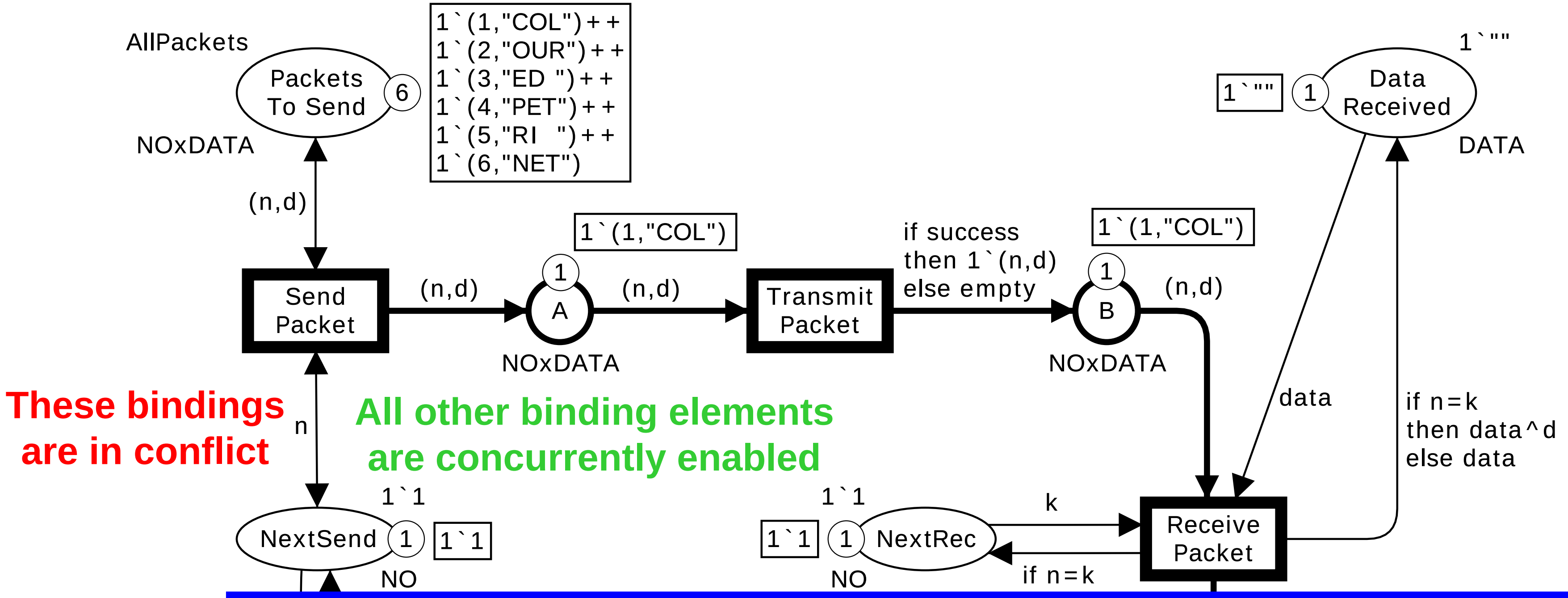


● SP = (SendPacket, <n=1, d="COL">)

● TP$^+$ = (TransmitPacket, <n=1, d="COL", success=true>)

TP$^-$ = (TransmitPacket, <n=1, d="COL", success=false>)

Three concurrent transitions

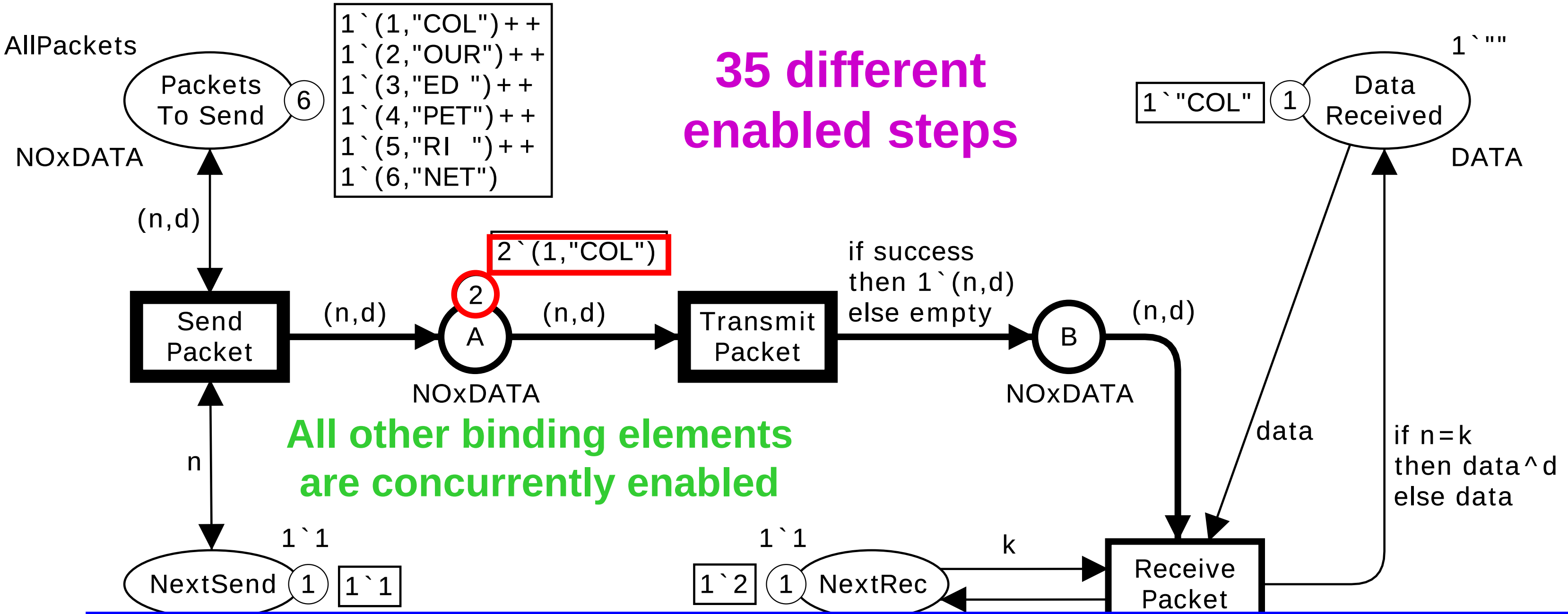


These bindings are in conflict

All other binding elements are concurrently enabled

- 🟢 SP = (SendPacket, <n=1, d="COL">)
- 🔴 TP^{+} = (TransmitPacket, <n=1, d="COL", success=true>)
- 🔴 TP^{-} = (TransmitPacket, <n=1, d="COL", success=false>)
- 🟢 RP = (ReceivePacket, <n=1, d="COL", k=1, data="">)

Three concurrent transitions



35 different enabled steps

All other binding elements are concurrently enabled

SP = (SendPacket,   <n=1, d="COL">)

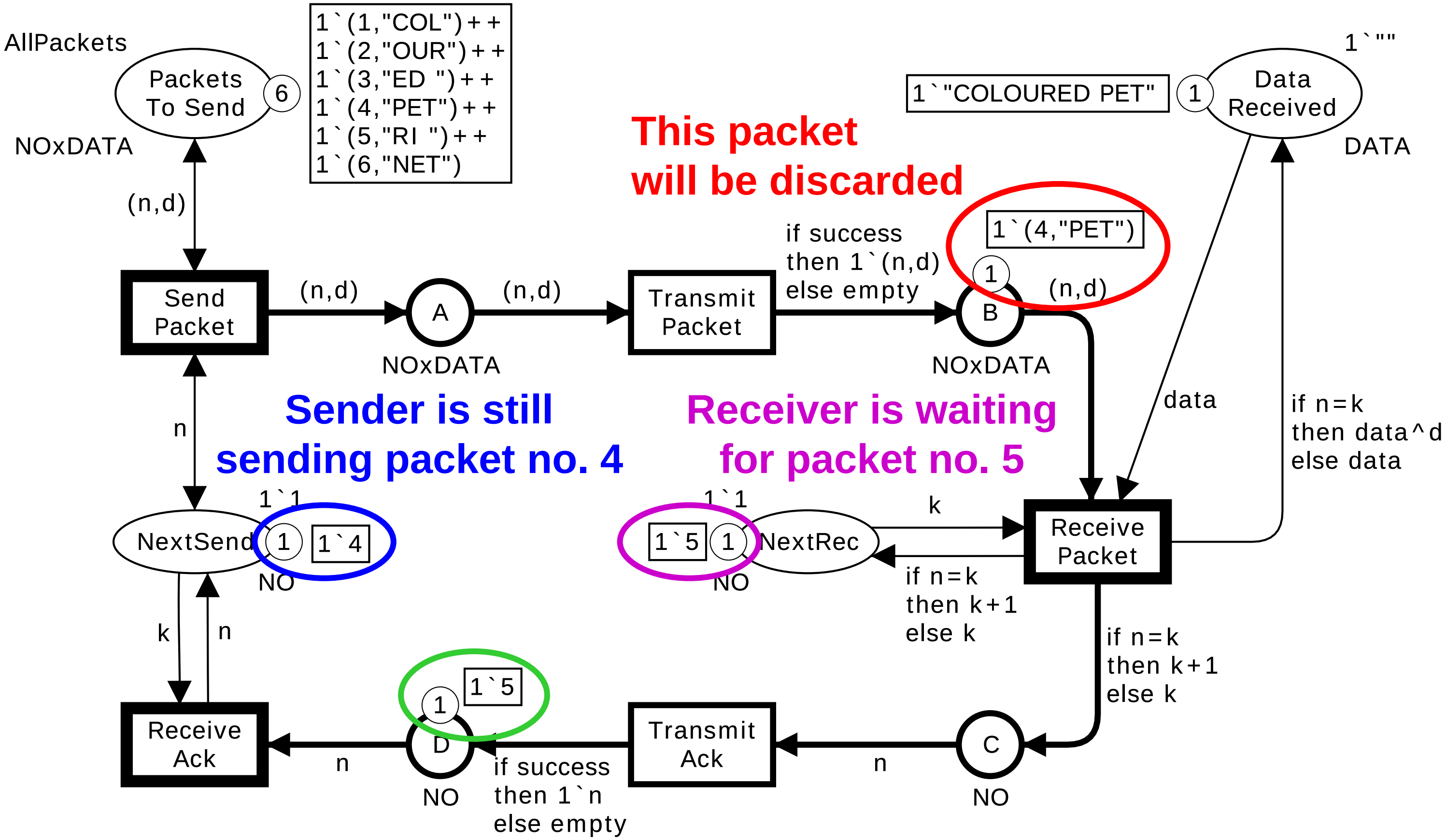
● TP^{+} = (TransmitPacket, <n=1, d="COL", success=true>)

● TP^{-} = (TransmitPacket, <n=1, d="COL", success=false>)

● TA^{+} = (TransmitAck,   <n=2, success=true>)

● TA^{-} = (TransmitAck,   <n=2, success=false>)

Possible marking after 50 steps



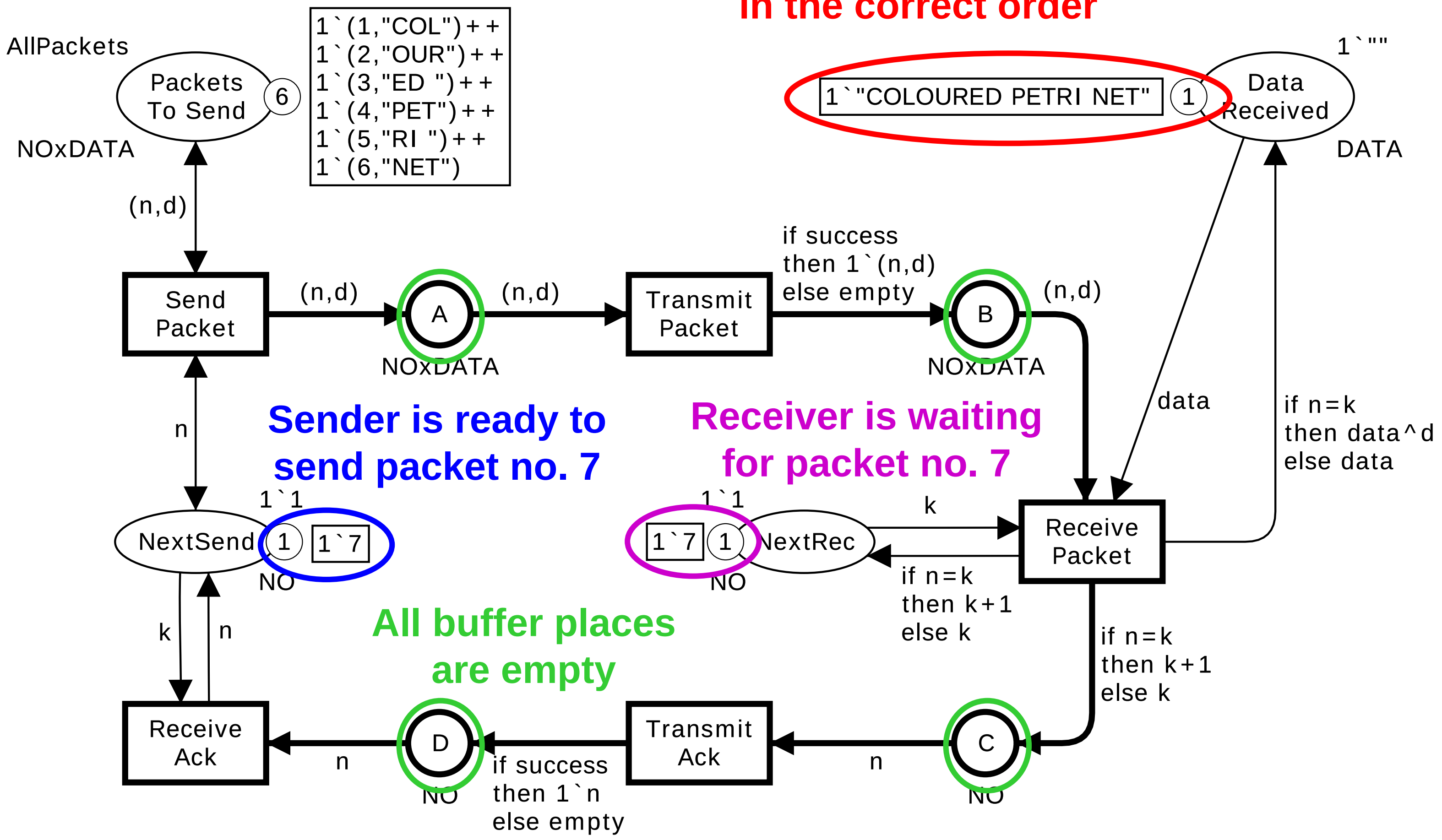
This packet will be discarded

Sender is still sending packet no. 4

Receiver is waiting for packet no. 5

An acknowledgement requesting packet no. 5 is arriving
When it is received the sender will start sending packet no. 5

Dead marking at the end of simulation



There is no packet no. 7

All packets have been received in the correct order

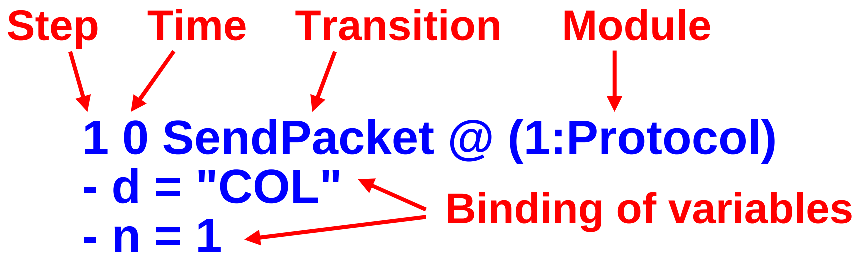
Sender is ready to send packet no. 7

Receiver is waiting for packet no. 7

All buffer places are empty

Simulation report

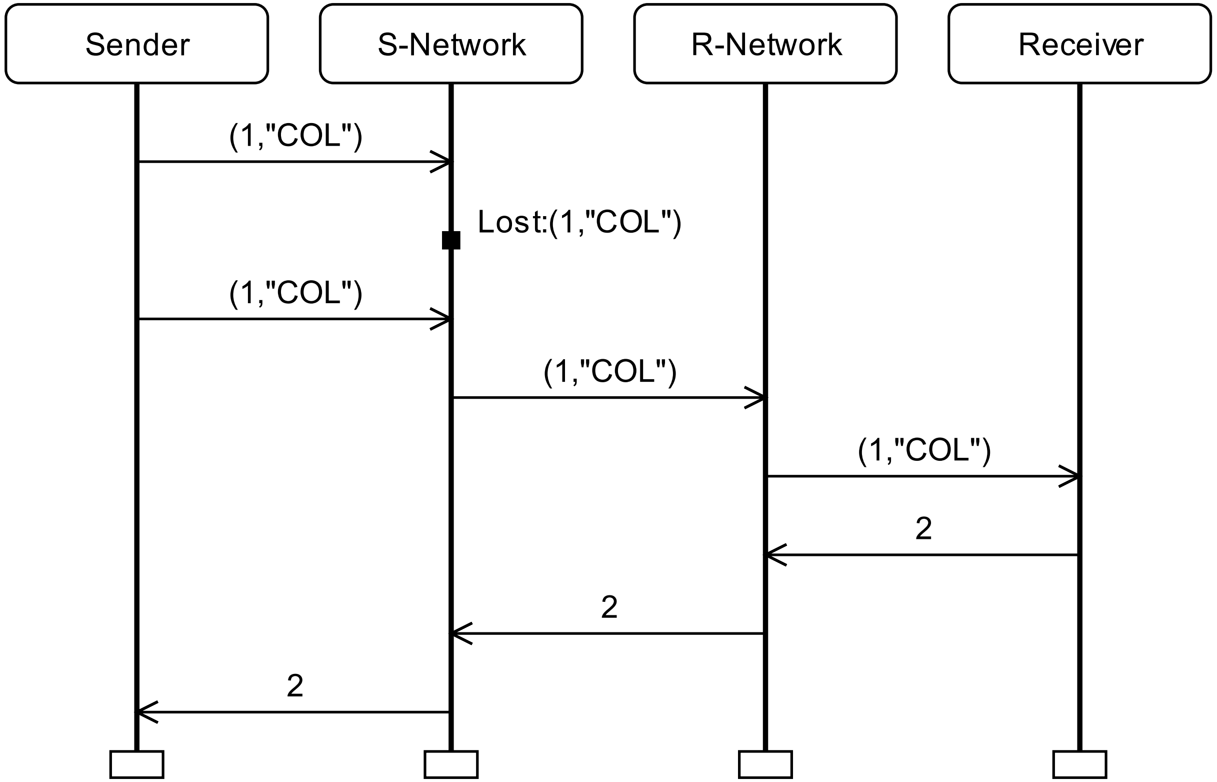
- Specifies the occurring transitions and their bindings.
- Automatically generated by the CPN Tools simulator.



```
1 0 SendPacket @ (1:Protocol)
- d = "COL"
- n = 1
2 0 TransmitPacket @ (1:Protocol)
- n = 1
- d = "COL"
- success = true
3 0 ReceivePacket @ (1:Protocol)
- k = 1
- data = ""
- n = 1
- d = "COL"
4 0 TransmitAck @ (1:Protocol)
- n = 2
- success = true
5 0 ReceiveAck @ (1:Protocol)
- k = 1
- n = 2
6 0 SendPacket @ (1:Protocol)
- d = "OUR"
- n = 2
```

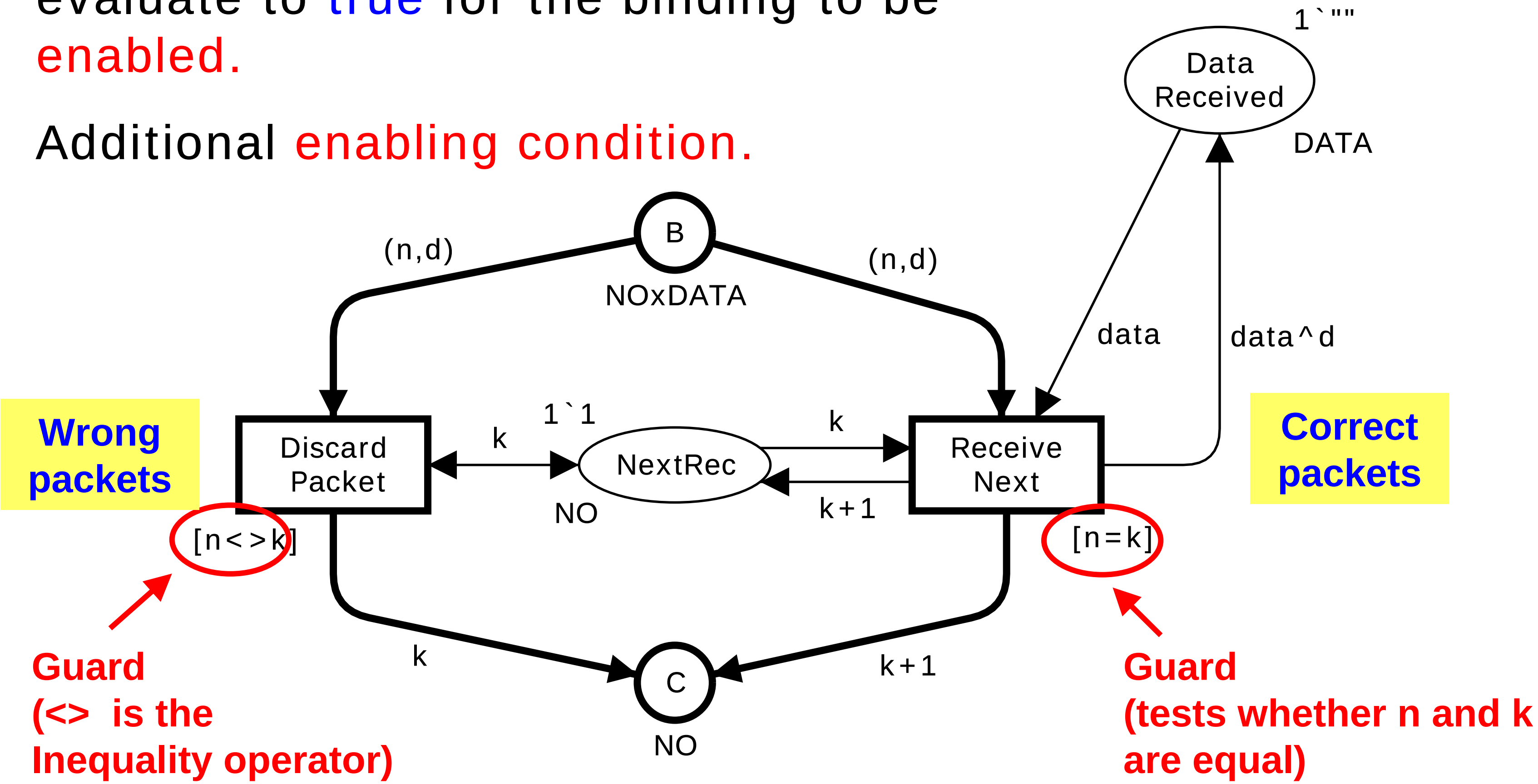
Message sequence chart

- Graphical high-level representation of occurrence sequence.
- Automatically generated by the CPN Tools simulator.
- Makes it easy to see what happened – also for non-CPN experts.



Transitions can have a guard

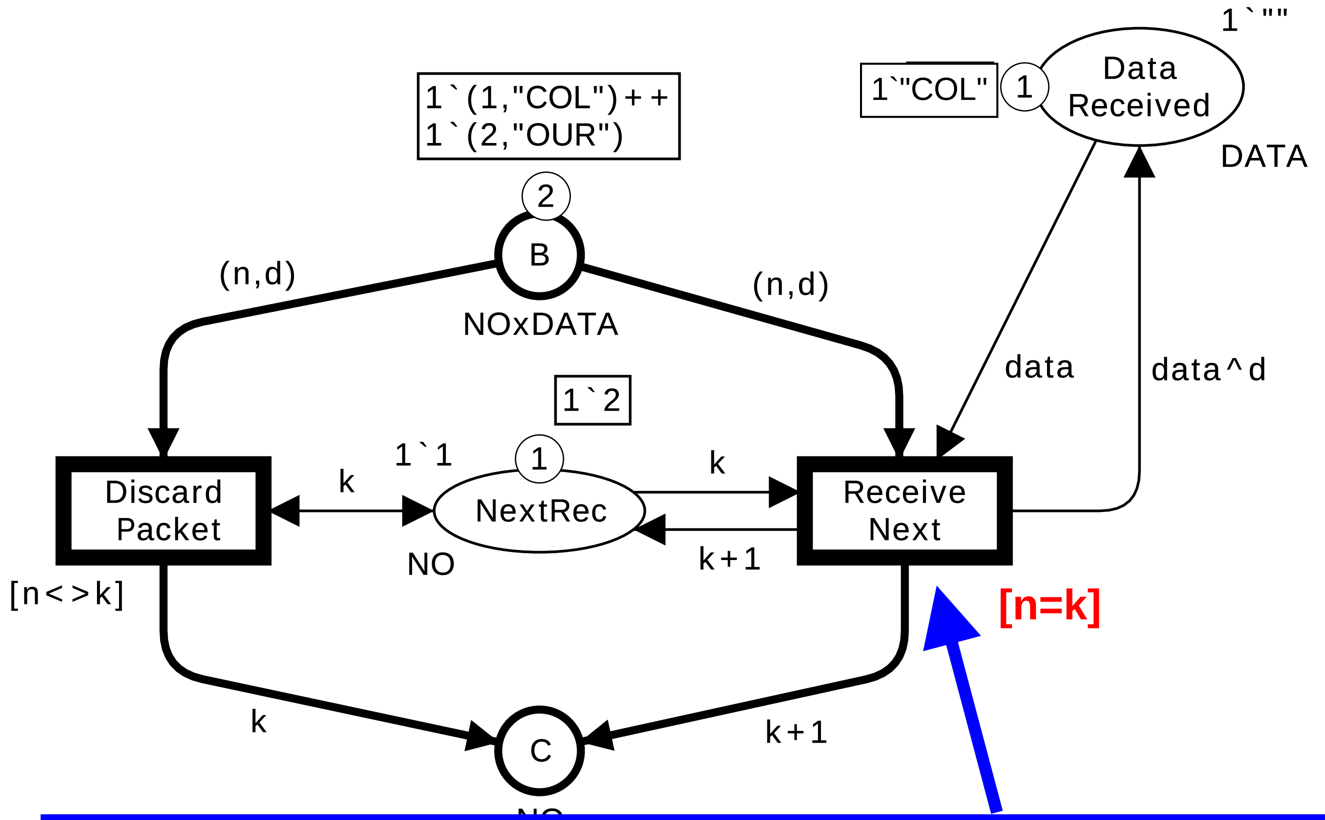
- Boolean expression, which must evaluate to true for the binding to be enabled.
- Additional enabling condition.



Guard
(<> is the
Inequality operator)

Guard
(tests whether n and k
are equal)

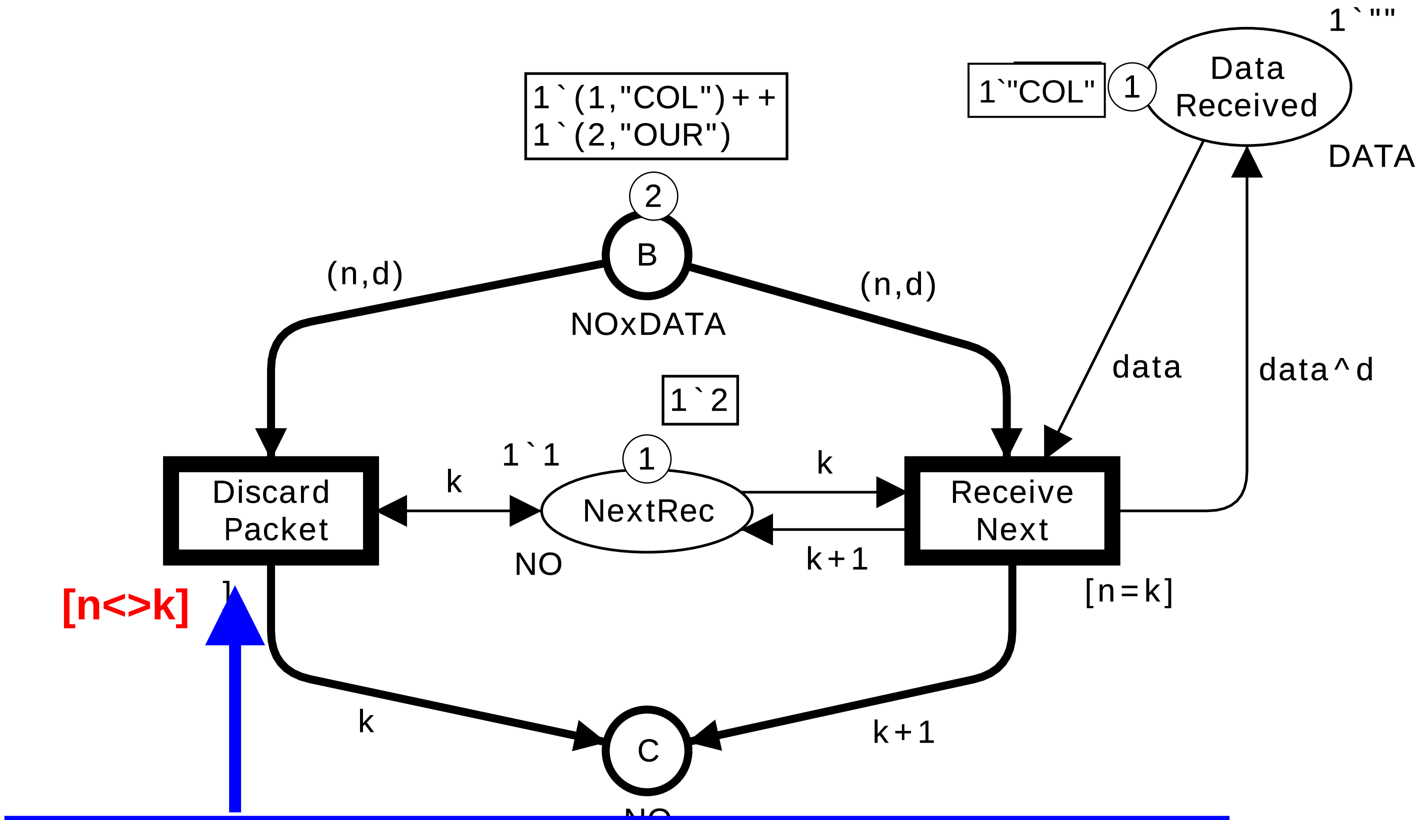
Guard must evaluate to true



false RN$_1$ = (ReceiveNext, <n=1, k=2, d="COL", data="COL">)

true 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")