U O W

Software Requirements, Specifications and Formal Methods

A/Prof. Lei Niu

UNIVERSITY
OF WOLLONGONG
AUSTRALIA

Coloured Petri Net

High-level Petri nets

- The invention of P/T-nets (Predicate/Transition nets) was the first step towards the kind of high-level Petri nets that we know today:
  - Tokens can be distinguished from each other and hence they are said to be coloured.
  - Transitions can occur in many different ways depending on the token colours of the available input tokens.
  - Arc expressions and guards are used to specify enabling conditions and the effects of transition occurrences.

High-level Petri nets

- The relationship between *CP-nets* and *ordinary Petri nets* (P/T-nets) is *analogous* to the relationship between *high-level programming languages* and *assembly code.*
  - In *theory,* the two levels have exactly the same *computational power.*
  - In *practice,* high-level languages have much more *modelling power* – because they have better structuring facilities, e.g., *types* and *modules.*
- Several other kinds of *high-level Petri Nets* exist. However, *Coloured Petri Nets* is the most widely used – in particular for *practical work.*

What is a Coloured Petri Net?

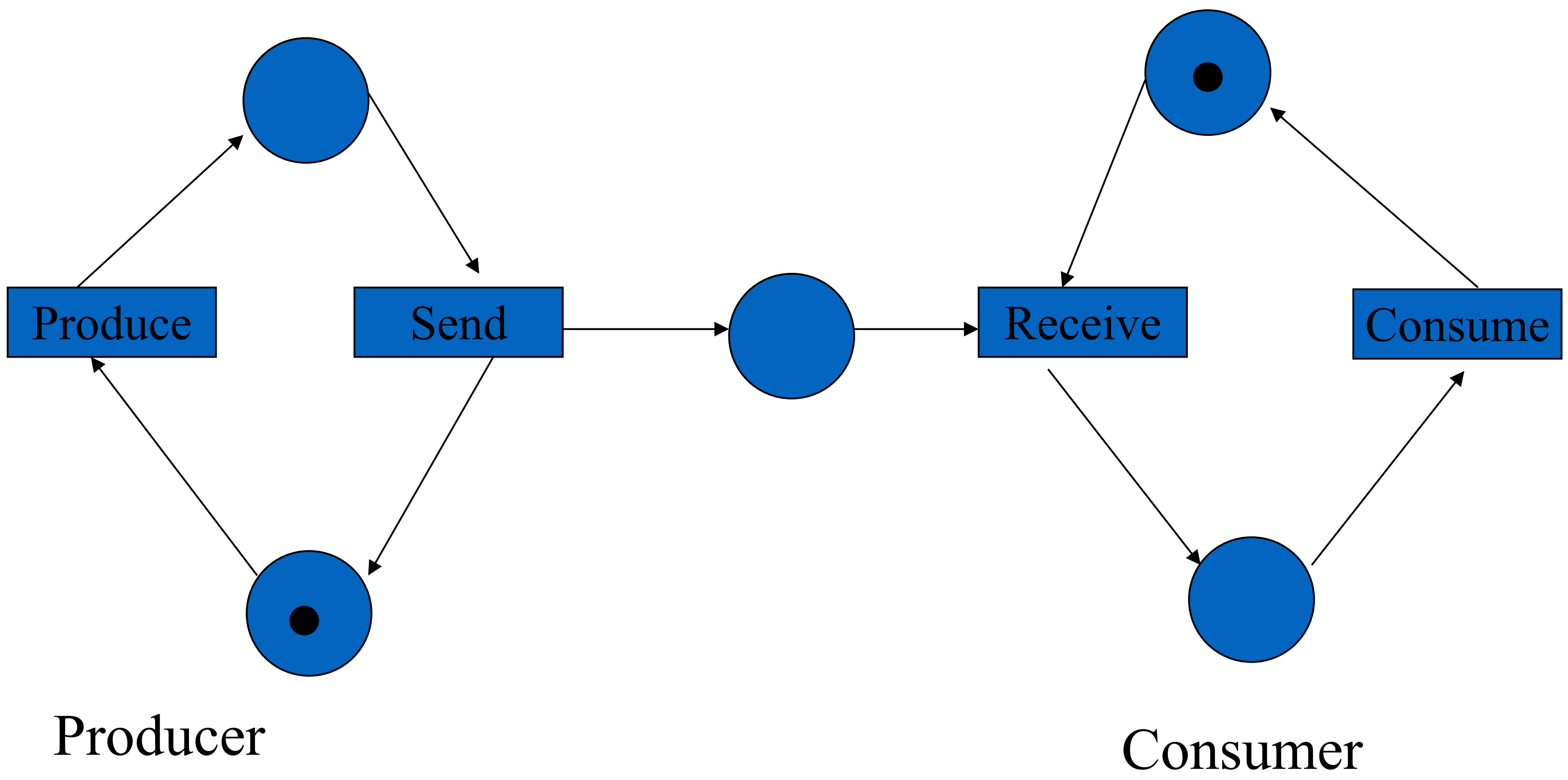
- *Modelling language* for systems where ***synchronisation***, ***communication***, and ***resource sharing*** are important.

- Combination of *Petri Nets* and *Programming Language.*
  - *Control structures*, *synchronisation*, *communication*, and *resource sharing* are described by *Petri Nets.*
  - *Data* and *data manipulations* are described by *functional programming language.*

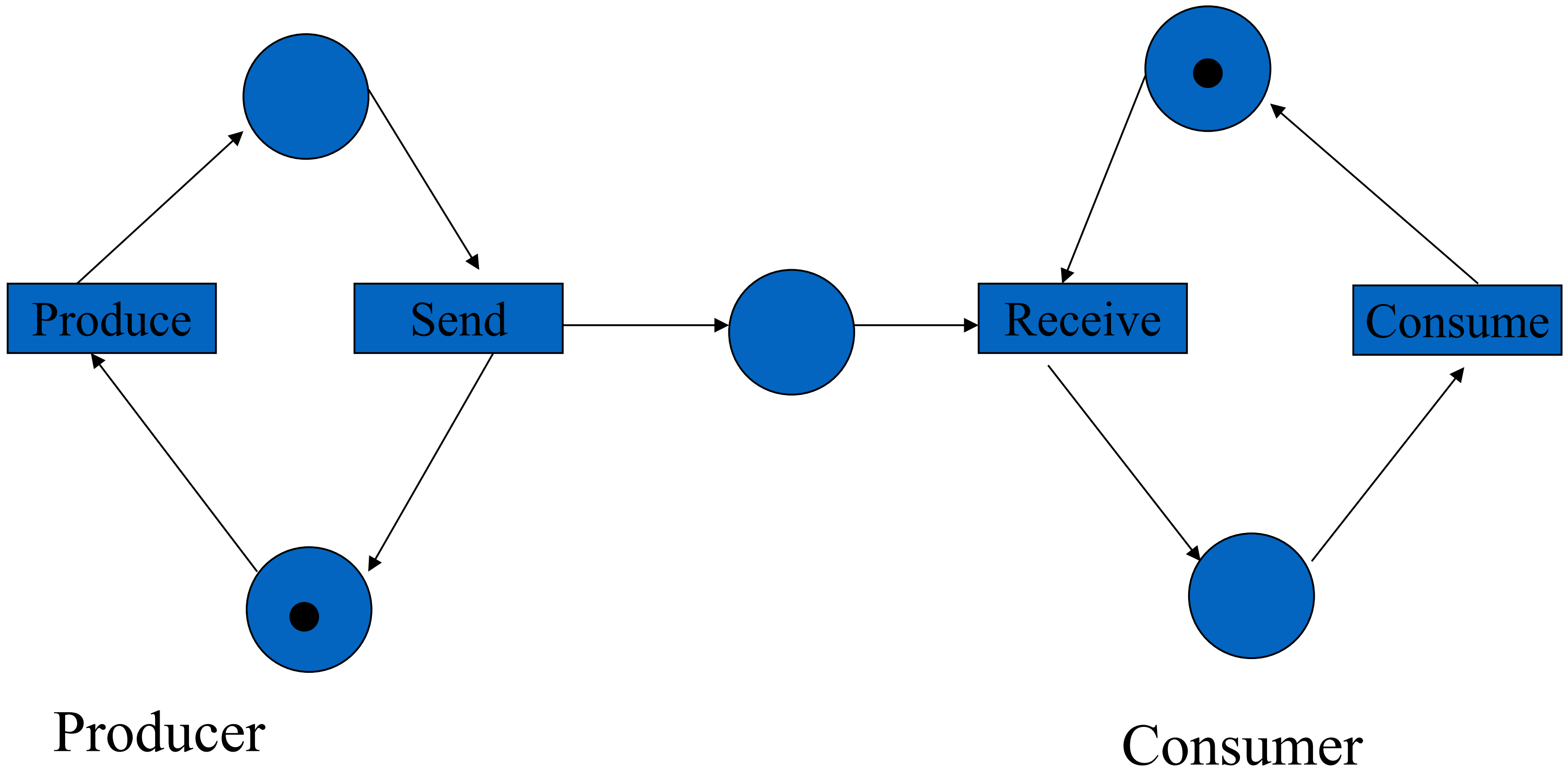
- CPN models are *validated* by means of *simulation* and *verified* by means of *state spaces* and *place invariants.*

- CPN models can be *executed* on computer.

Opening Question: Producer/Consumer Problem

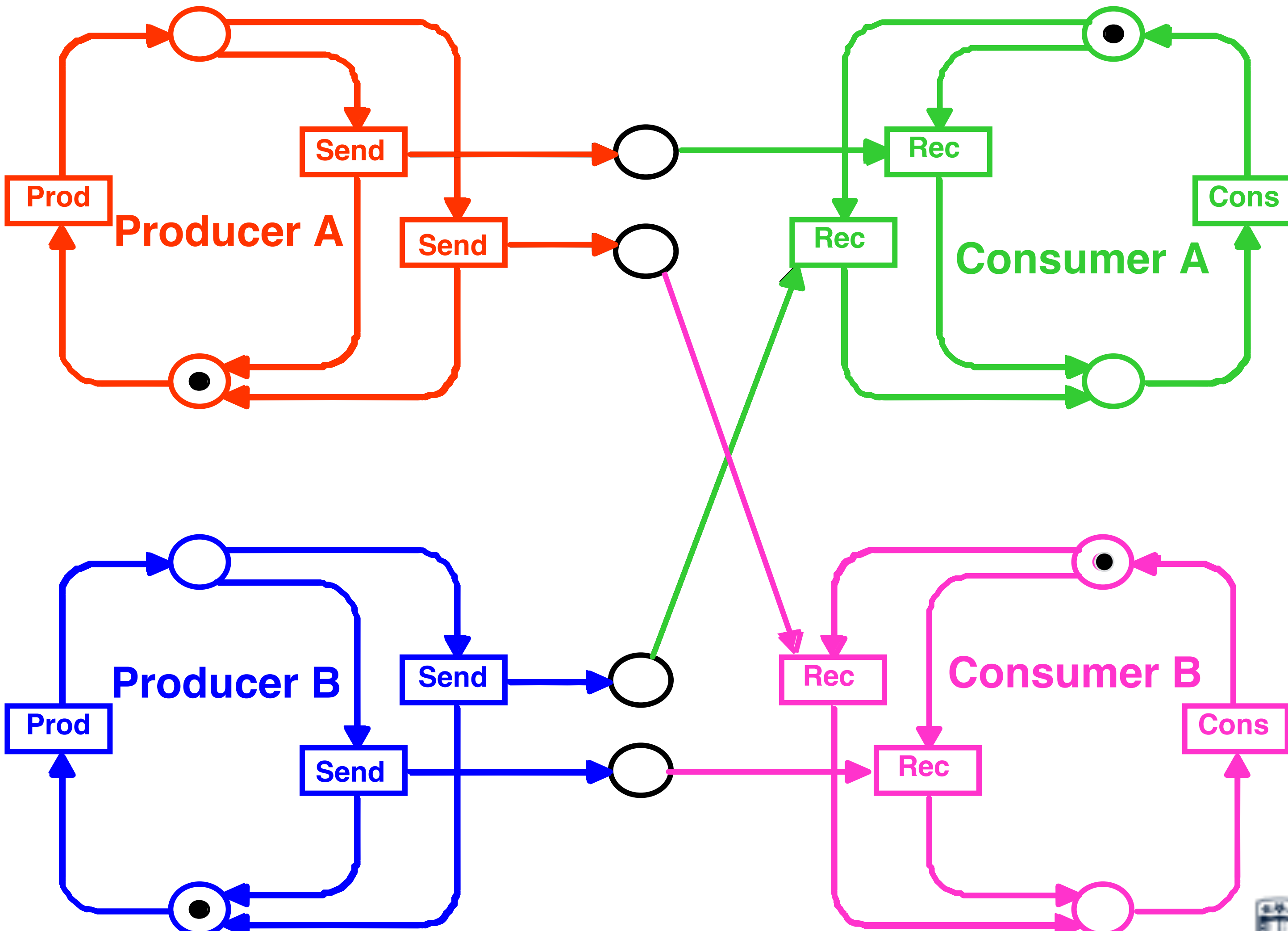


Opening Question: Producer/Consumer Problem



How to model "many-to-many" scenarios?

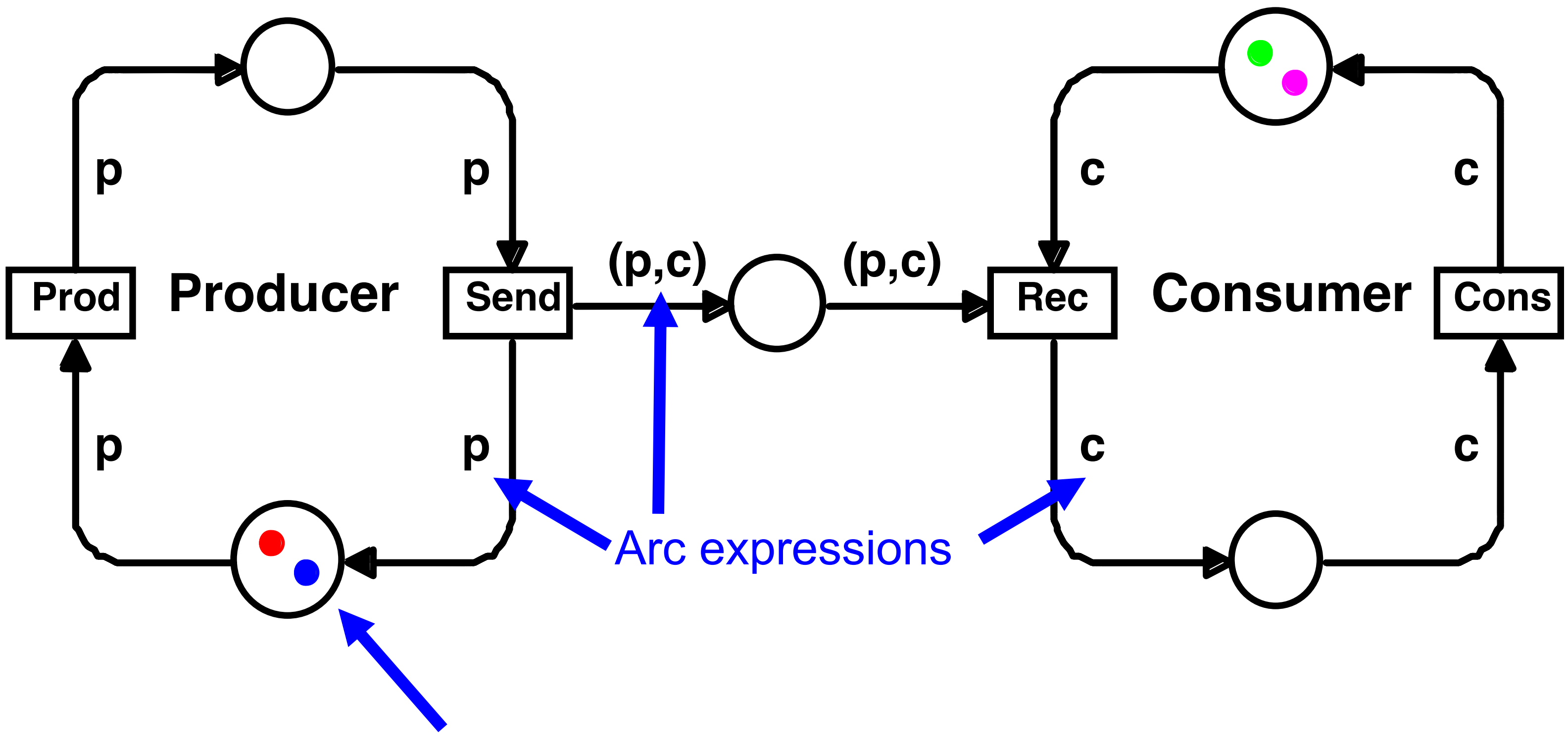
Repeated net structure



High-level Petri Net (P/T net)

D = { red, blue, green, purple }

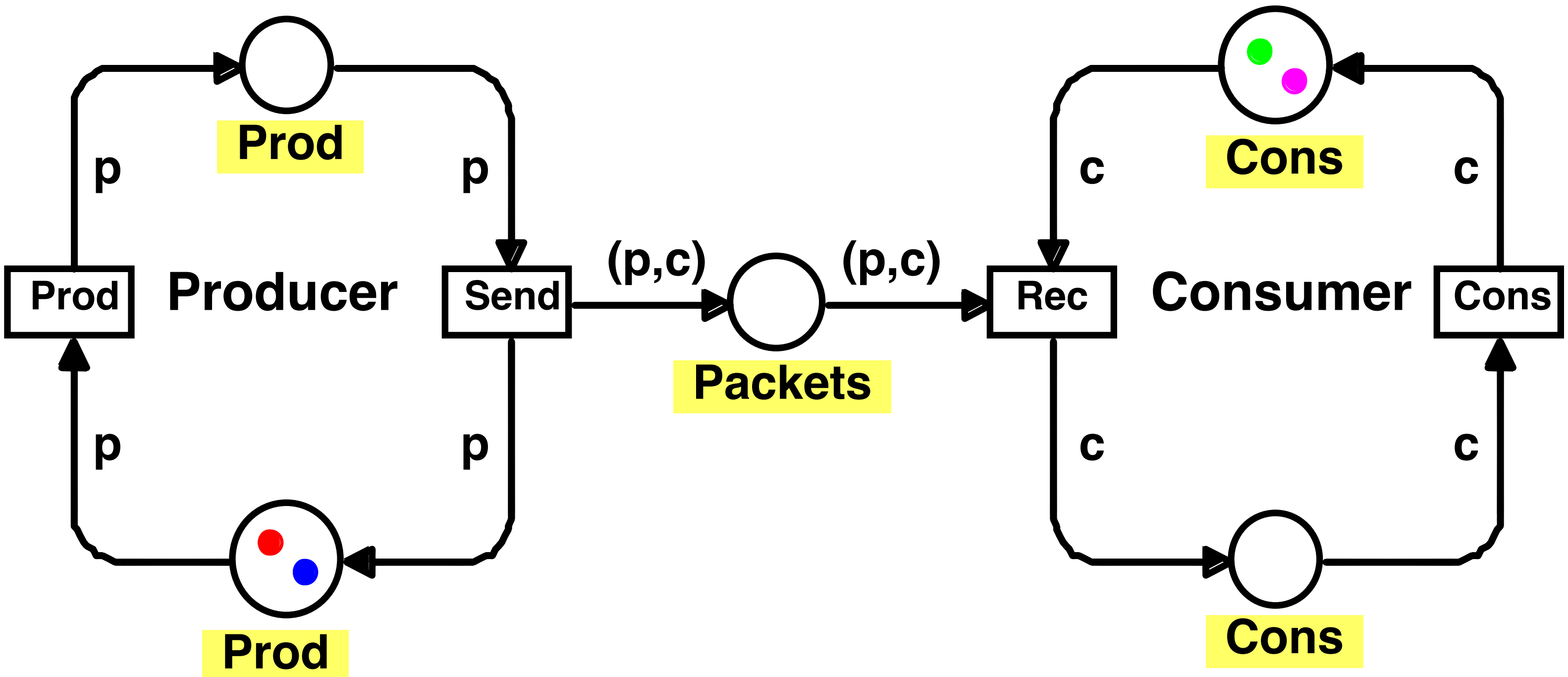
var p,c : D



Each token carries a data value

It is coloured !!!

Coloured Petri Nets



colset Prod = { red, blue }

colset Cons = { green, purple }

colset Packets = product Prod * Cons

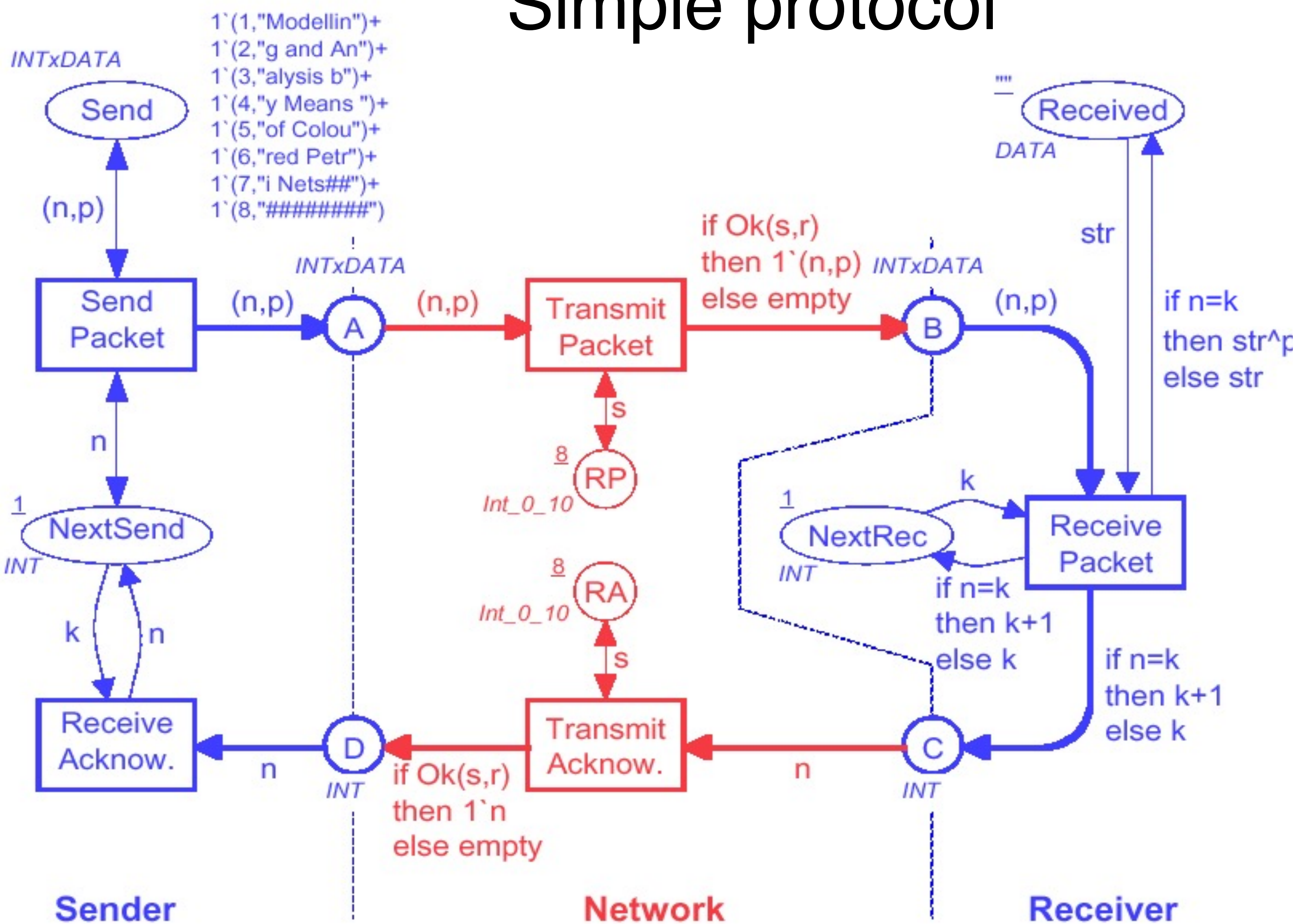
var p : Prod

var c : Cons

Colour sets = Types

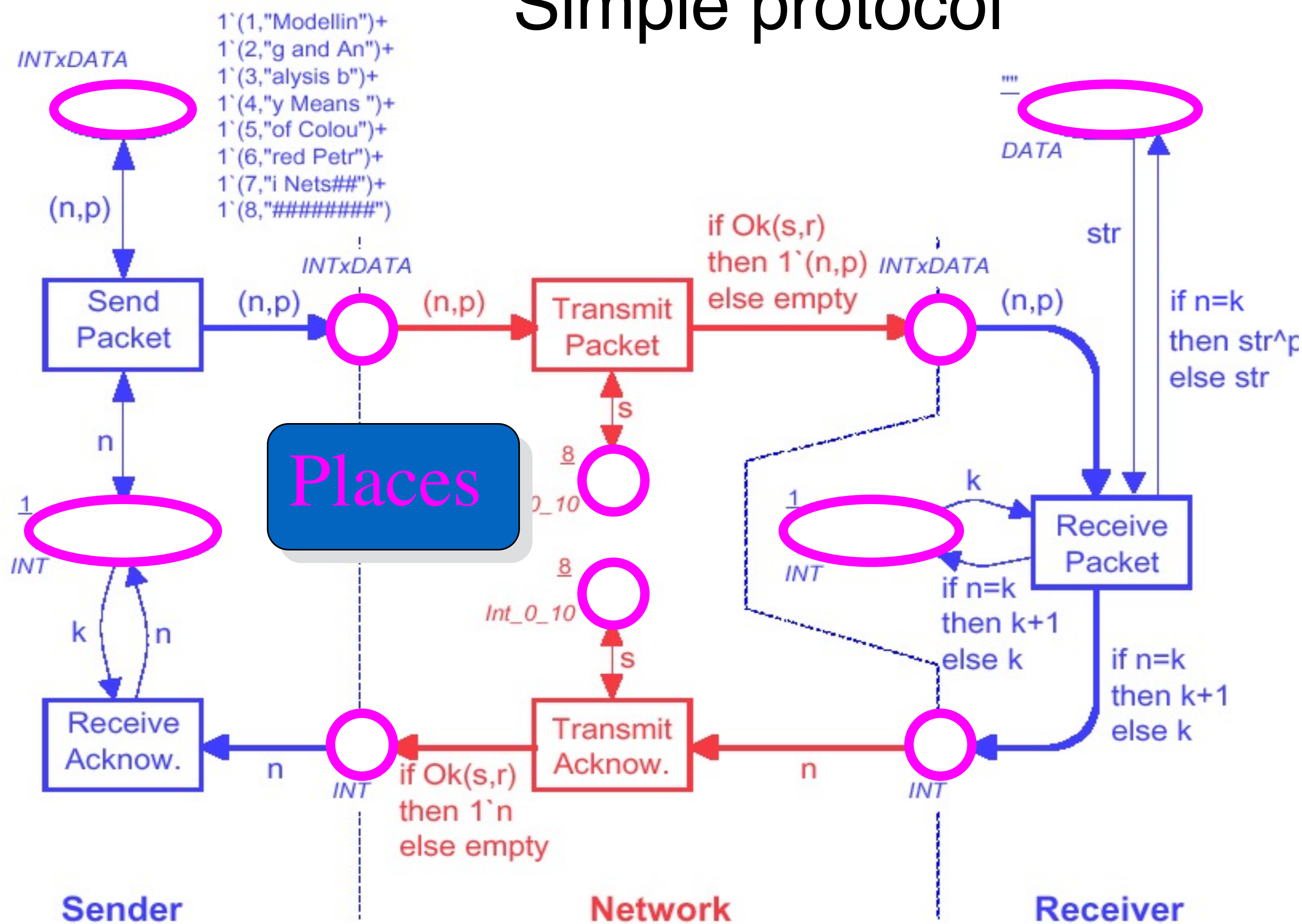
- We use data types to specify the kinds of tokens which we allow on the individual places.
- Types can be arbitrarily complex:
  - *Atomic* (e.g., integers, strings, Booleans and enumerations).
  - *Structured* (e.g., products, records, unions, lists, and subsets).
- The use of types allows us to make more readable descriptions with mnemonics type names such as:
  - **PROD, CONS, PACKETS**
- We also get more correct descriptions.
  - Automatic type checking of arc expressions.

Simple protocol

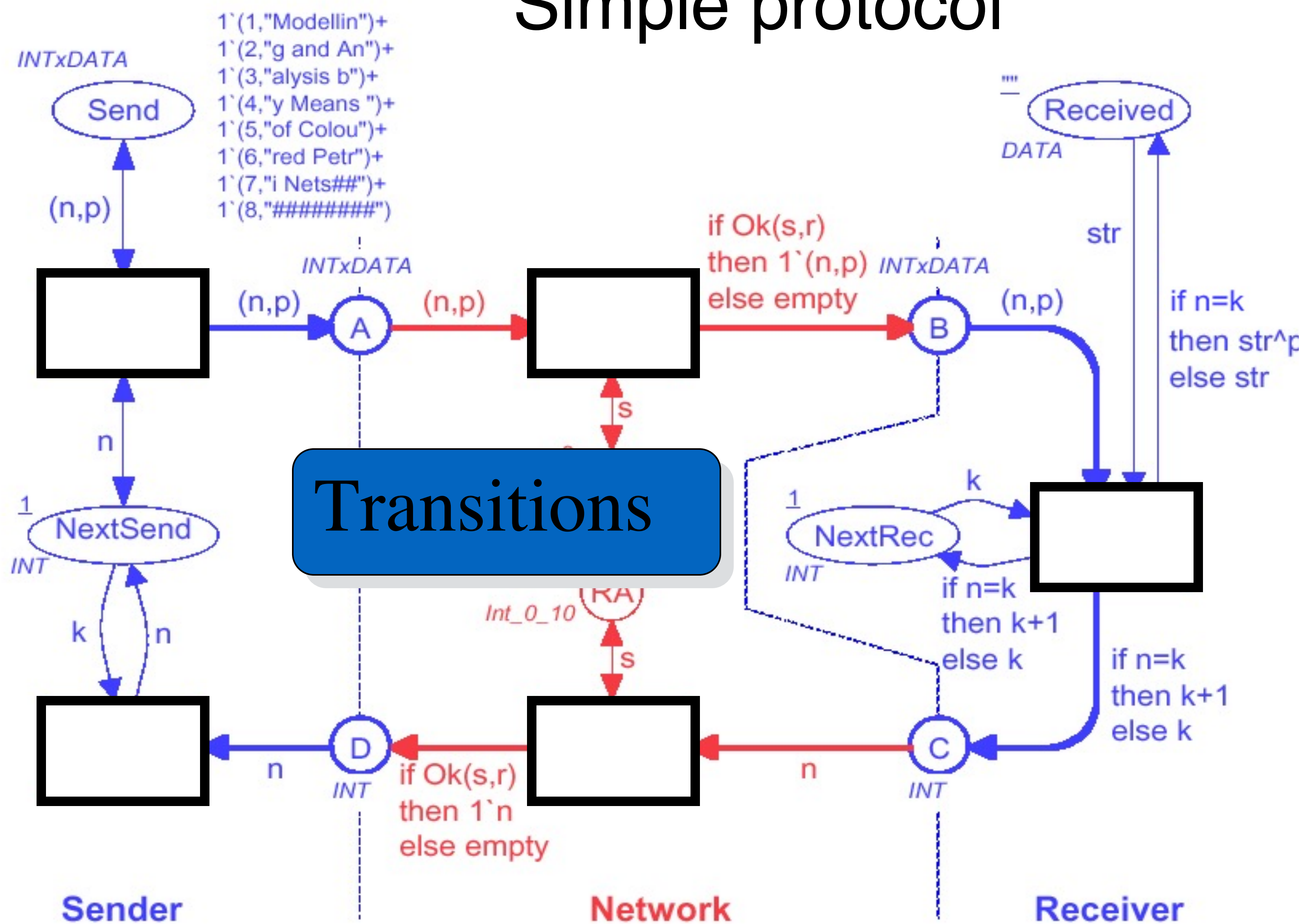


UNIVERSITY OF WOLLONGONG AUSTRALIA

Simple protocol

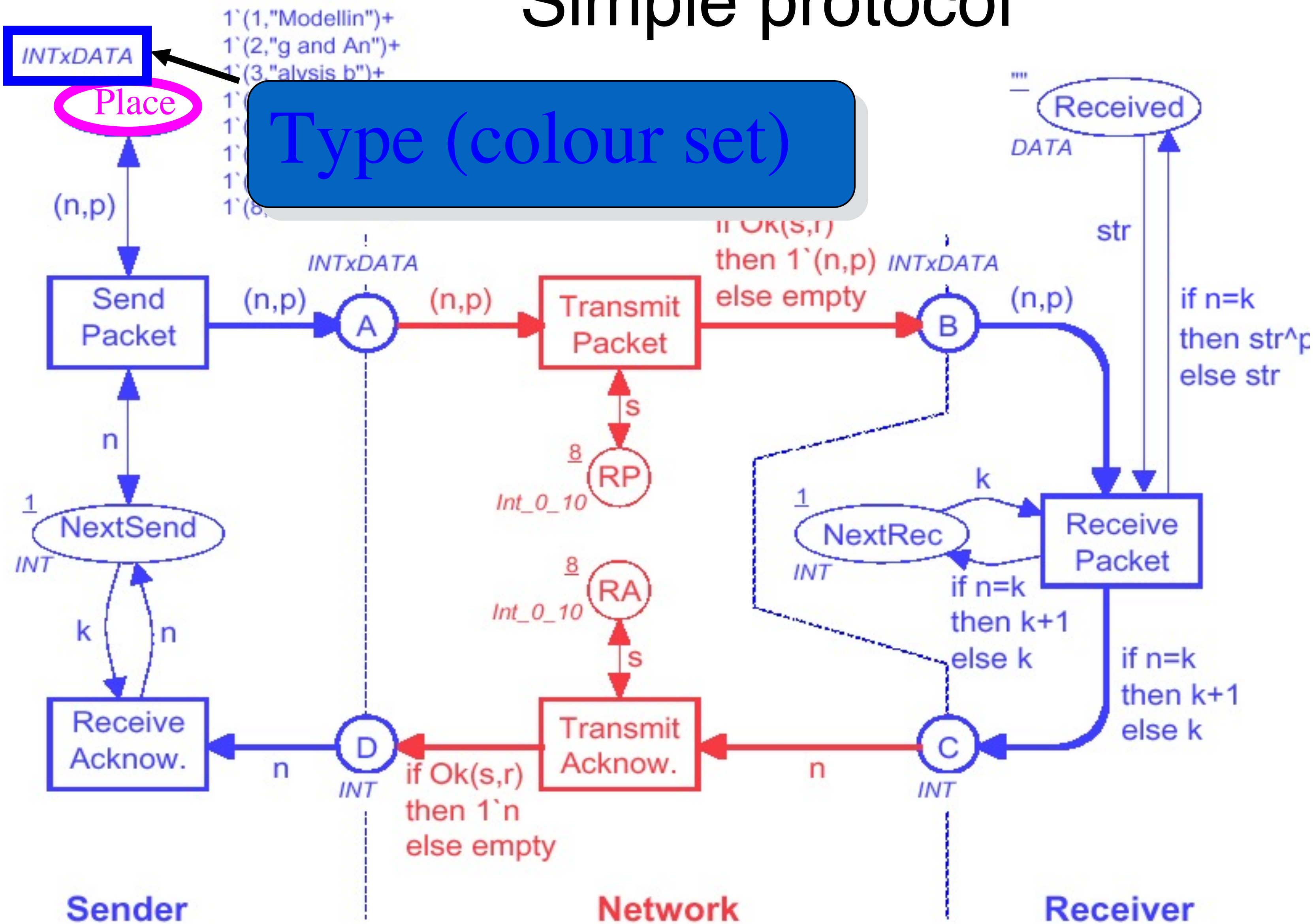


Simple protocol



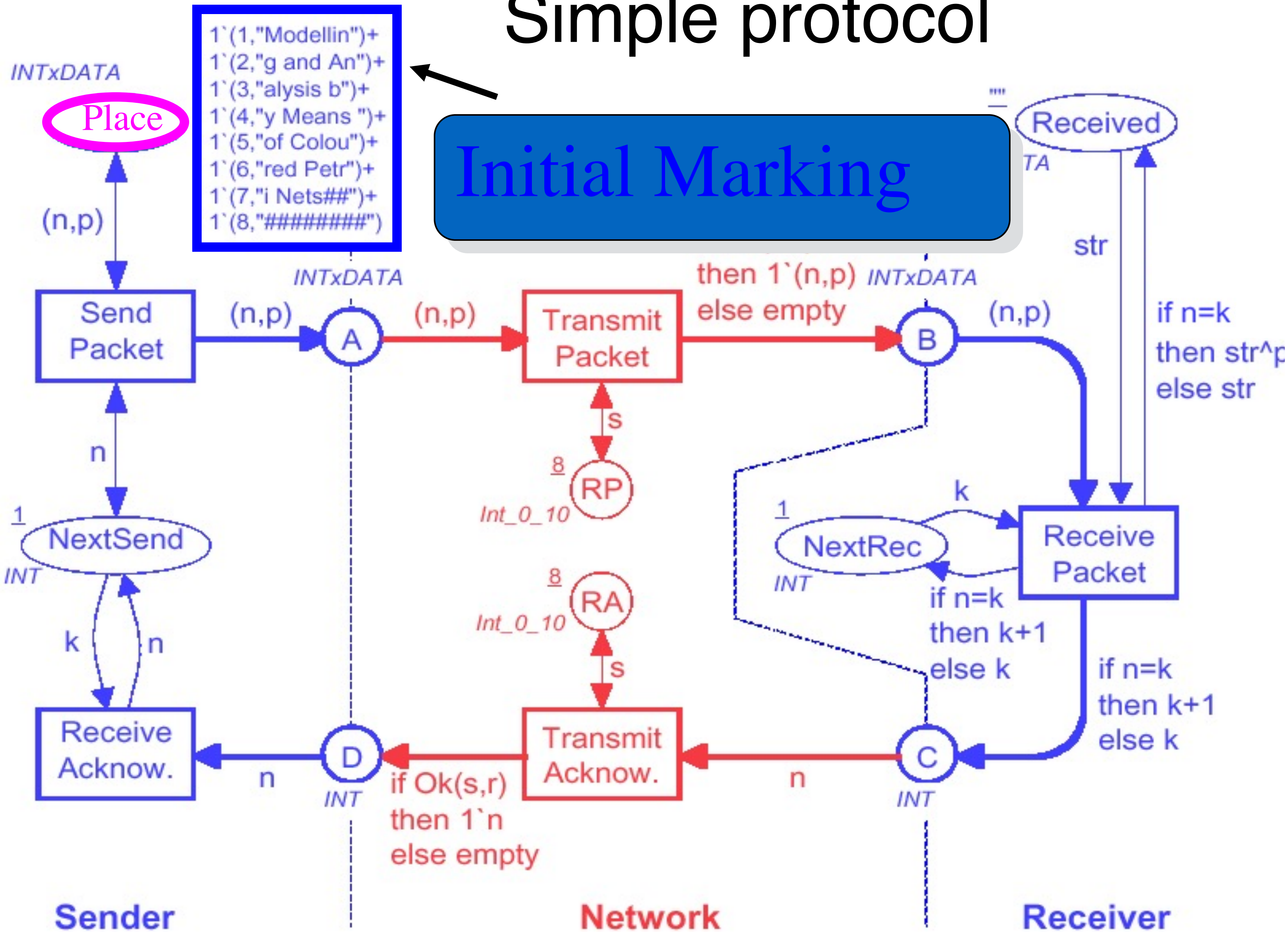
UNIVERSITY OF WOLLONGONG AUSTRALIA

Simple protocol

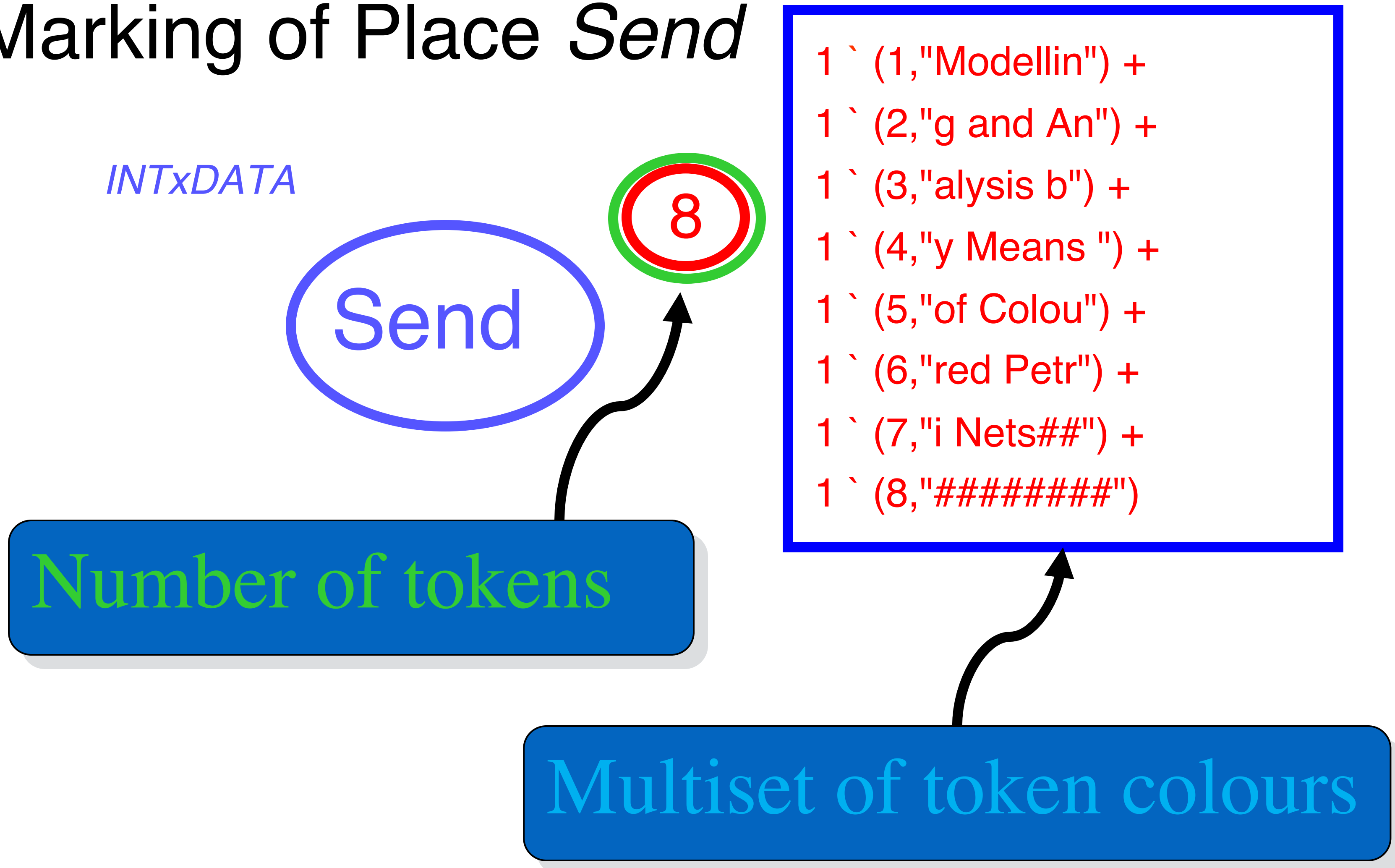


UNIVERSITY OF WOLLONGONG AUSTRALIA

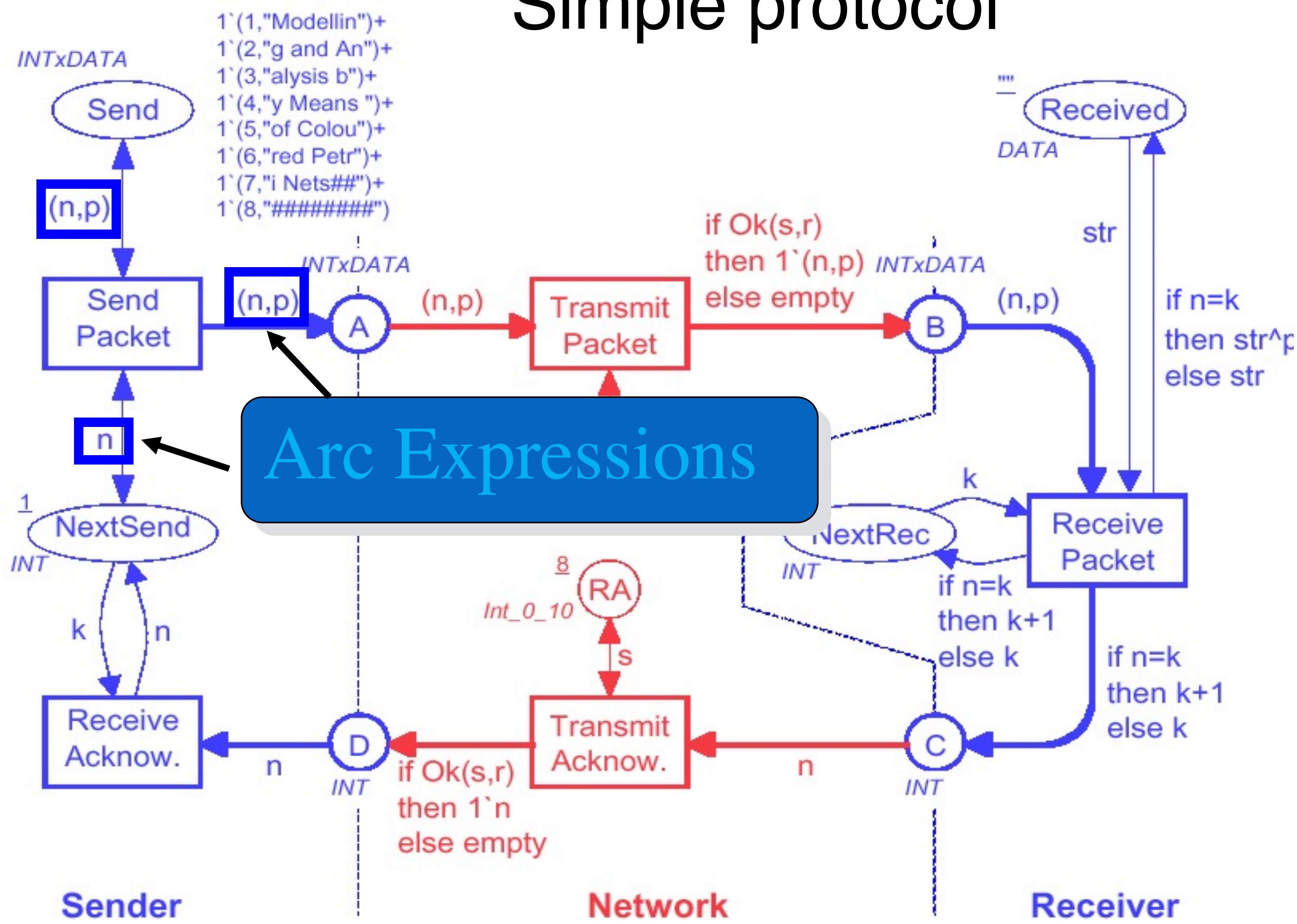
Simple protocol



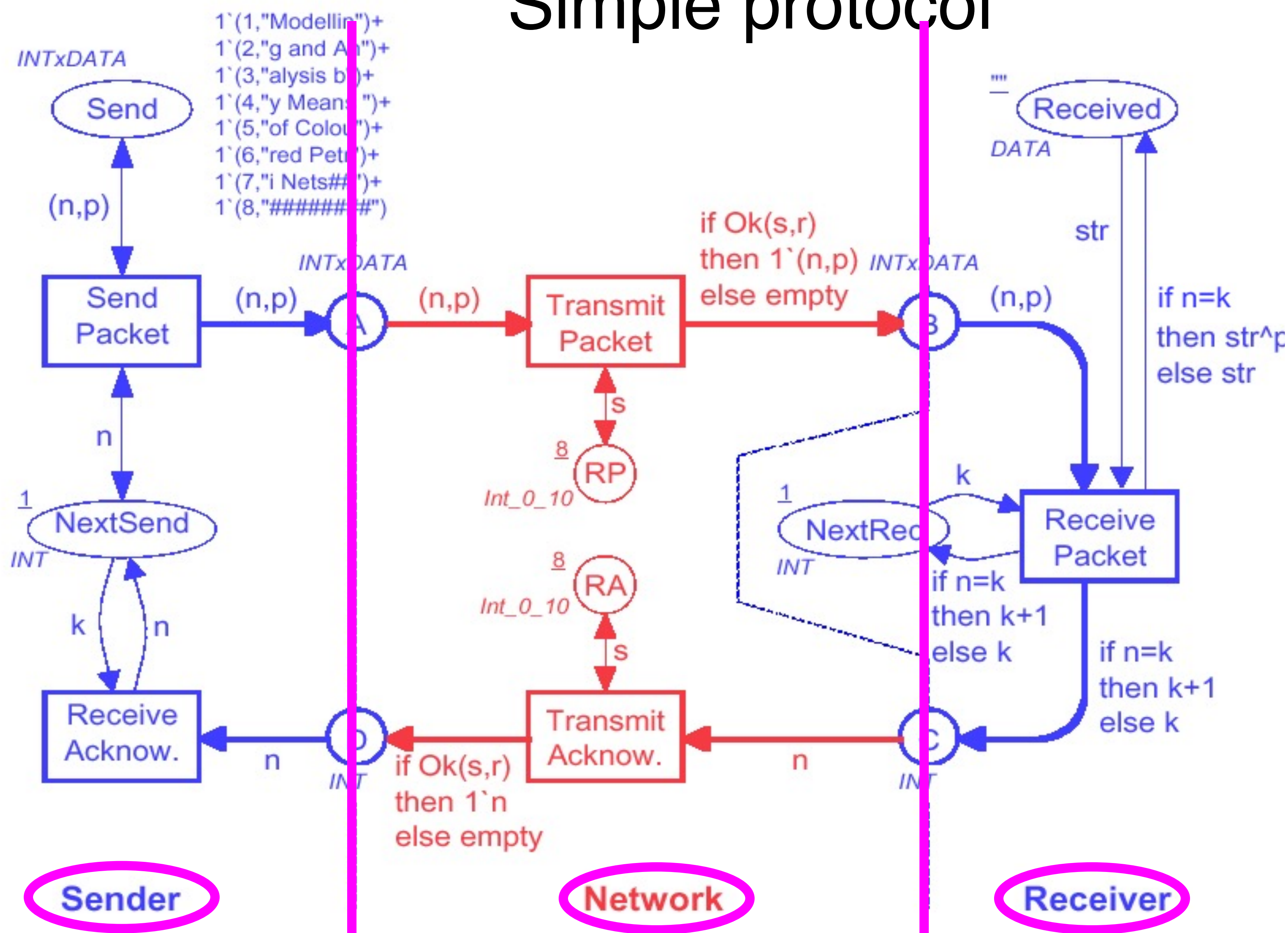
Marking of Place *Send*



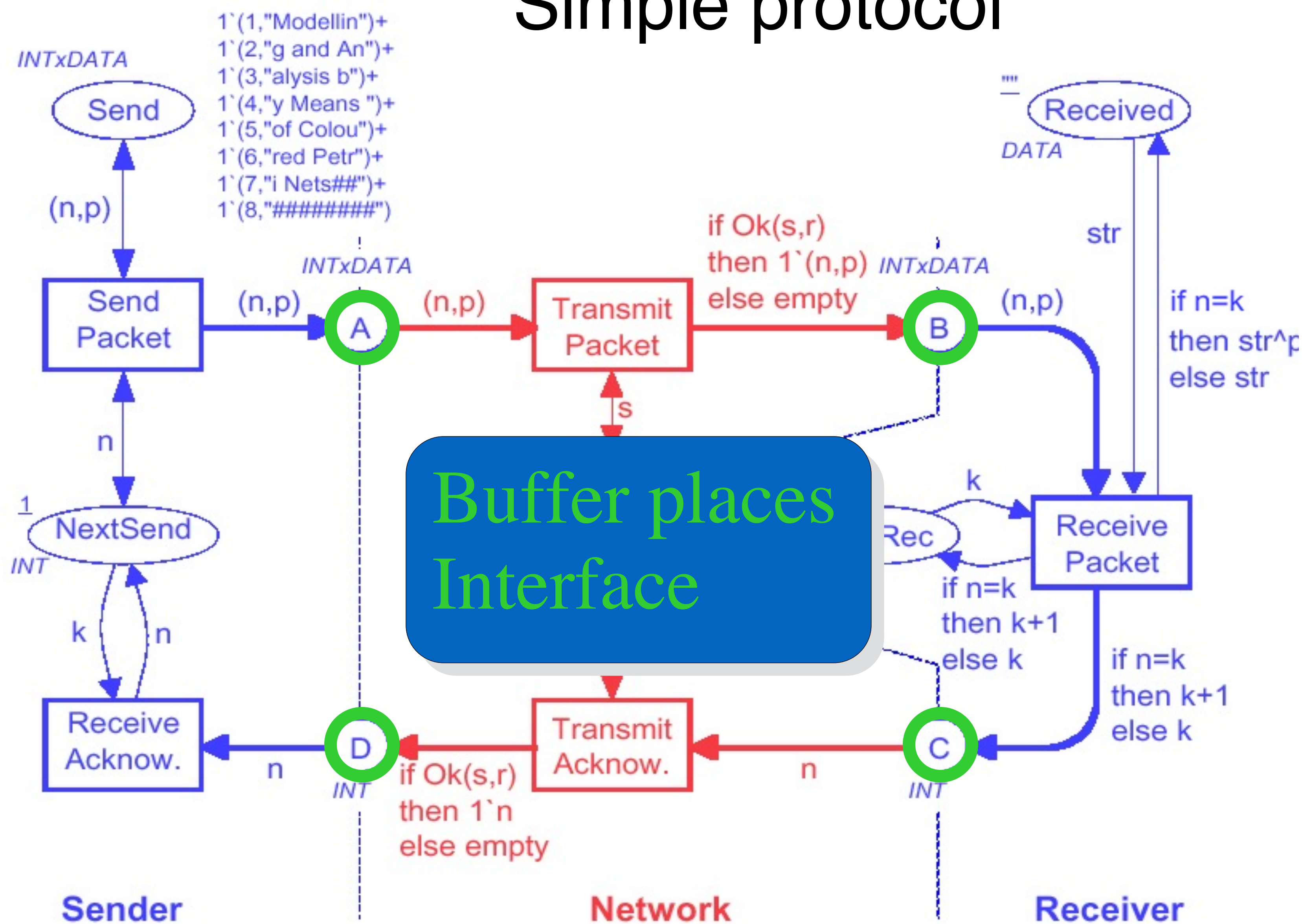
Simple protocol



Simple protocol

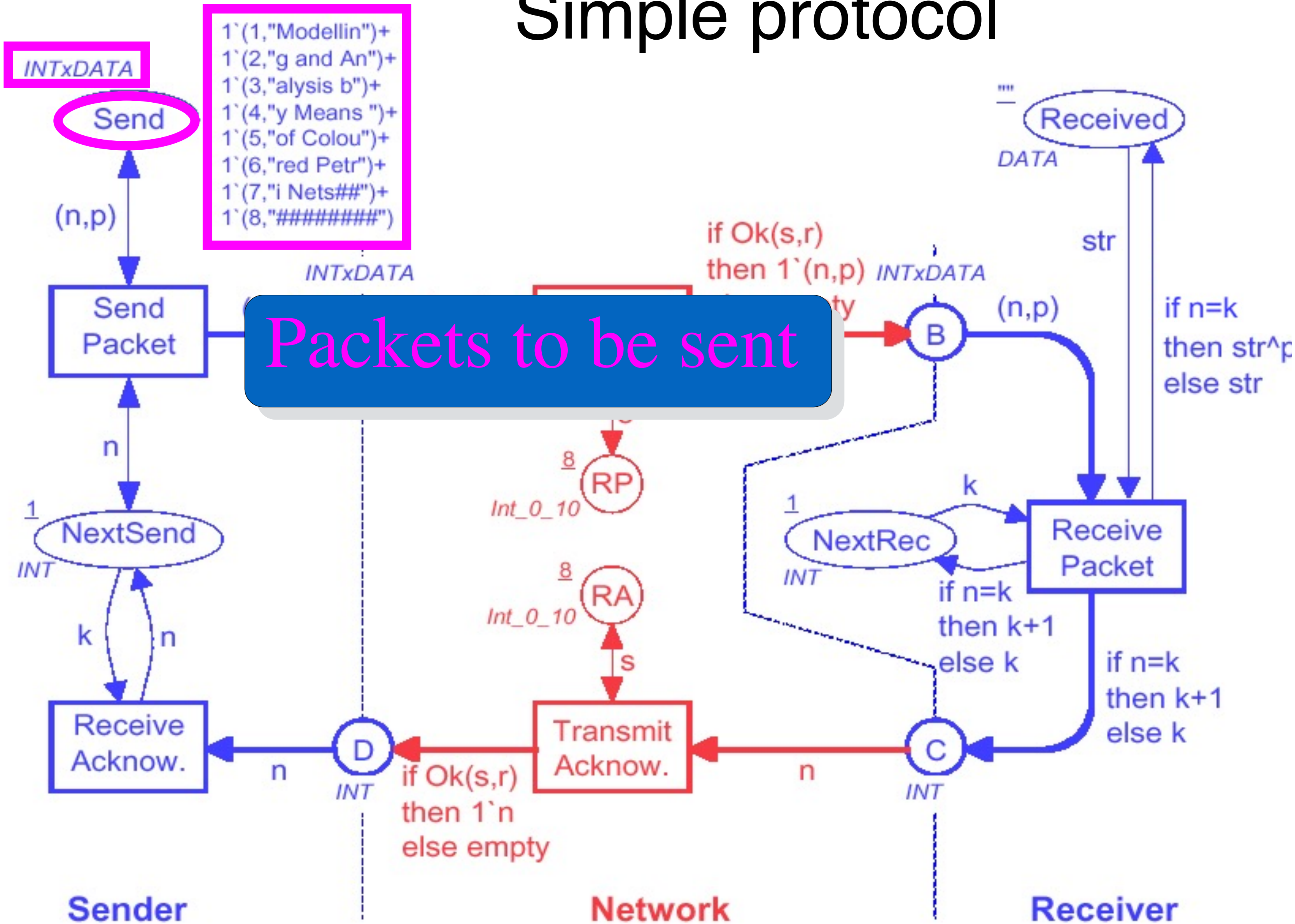


Simple protocol



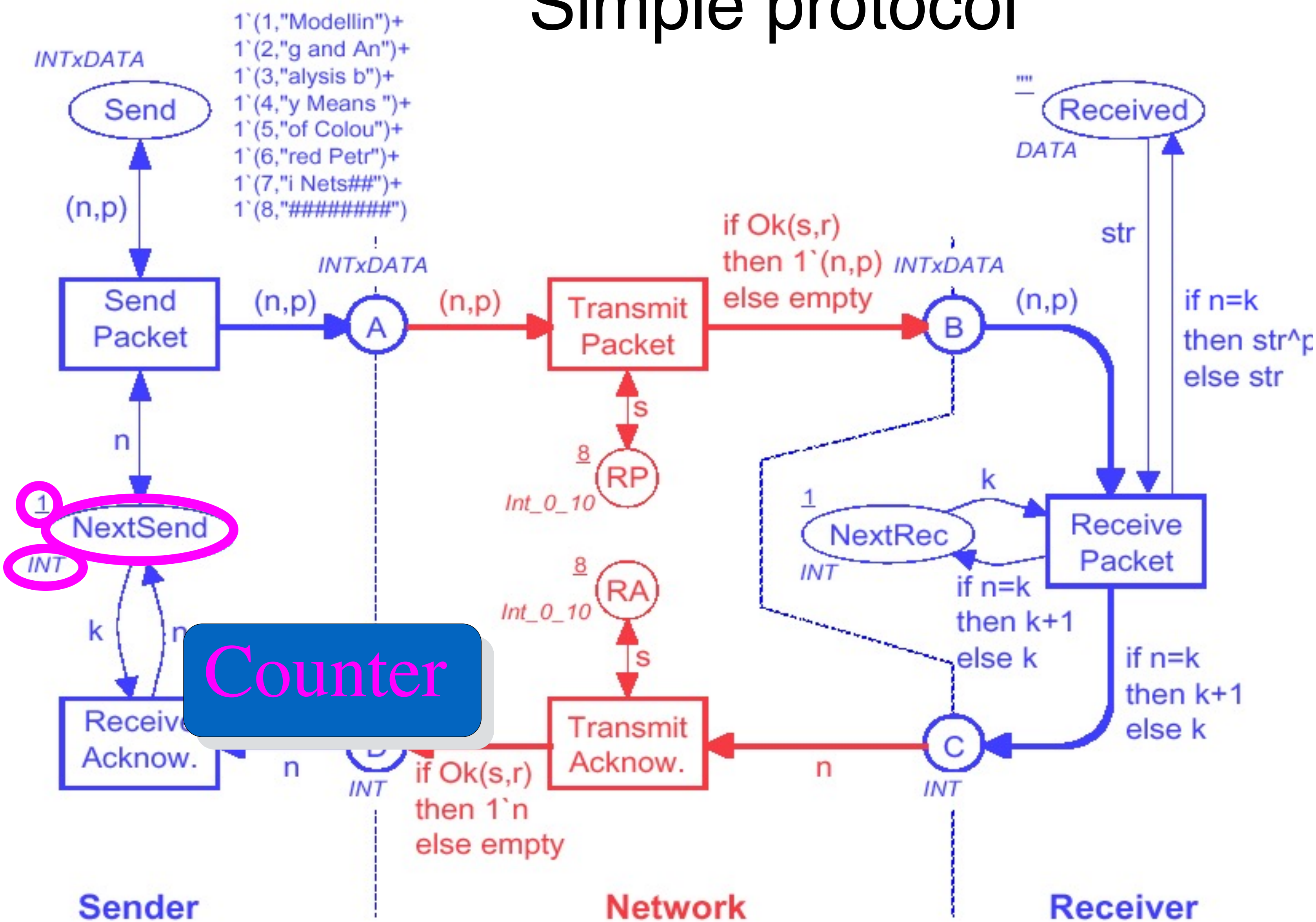
UNIVERSITY OF WOLLONGONG AUSTRALIA

Simple protocol



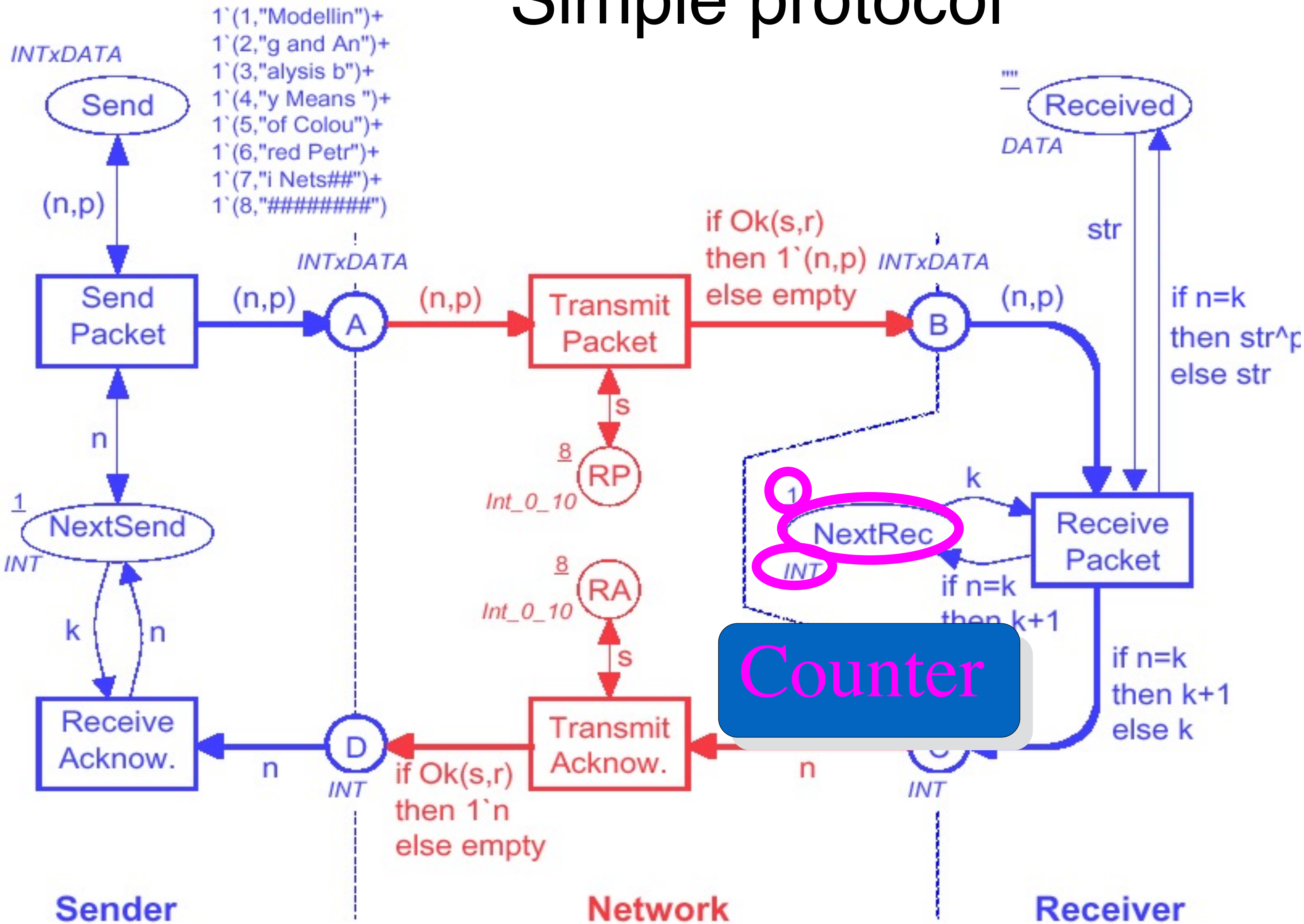
Packets to be sent

Simple protocol

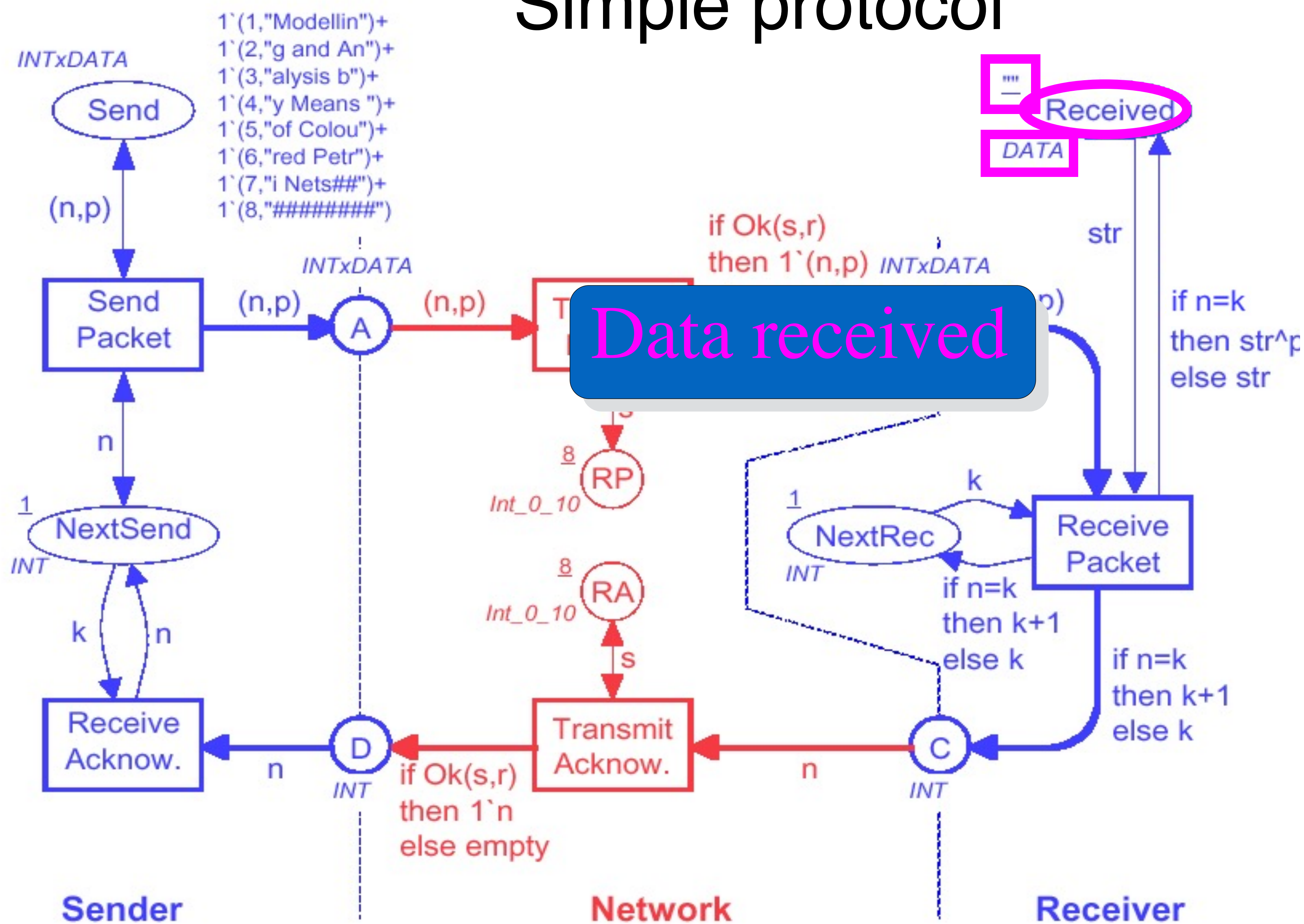


UNIVERSITY OF WOLLONGONG AUSTRALIA

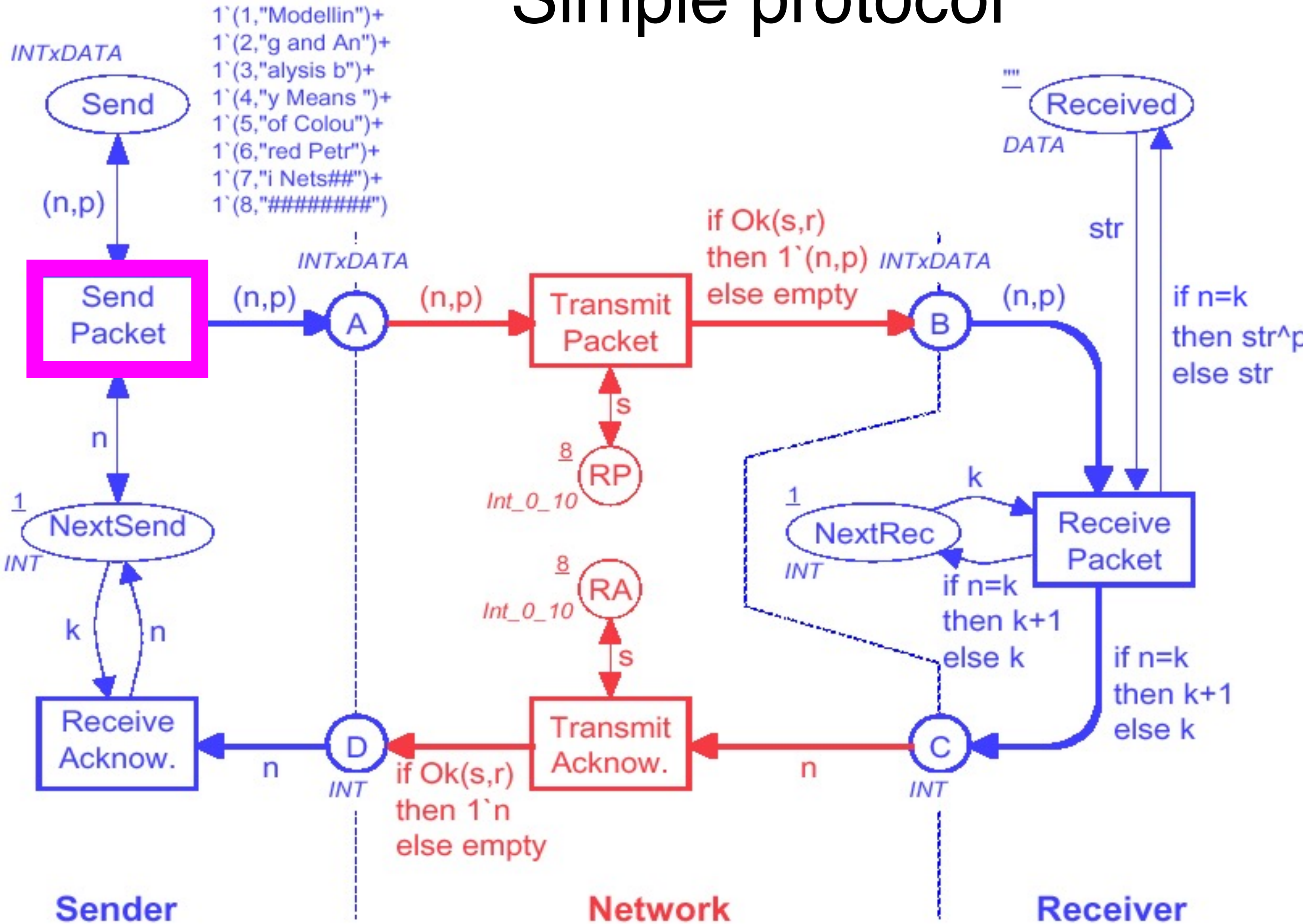
Simple protocol



Simple protocol



Simple protocol



Send packet

p = "Modellin"

- The binding (i.e., variable assignment)

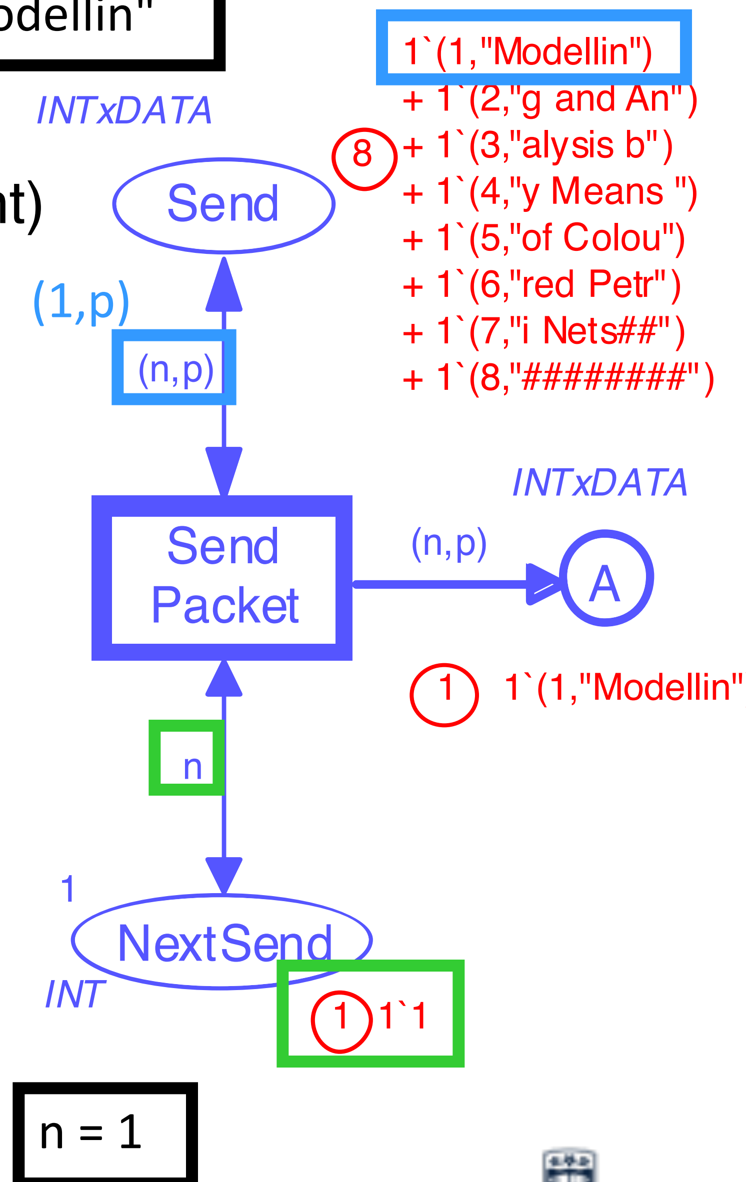
<n=1,p="Modellin">

is *enabled.*

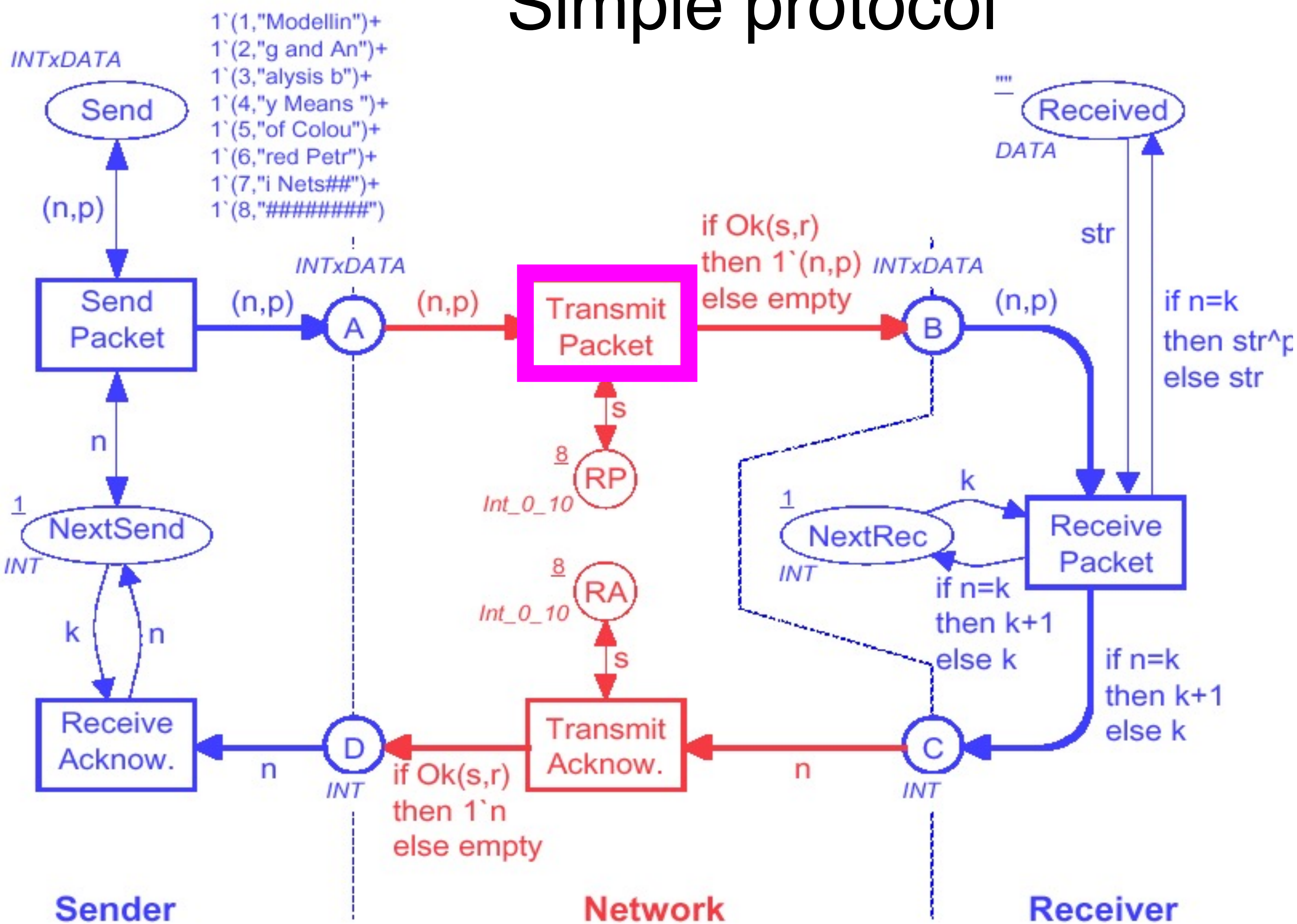
When the binding *occurs* it *adds a token* to place A.

This represents that the packet (1,"Modellin") is *sent to the network.*

The packet is *not removed* from place *Send* and the *NextSend* counter is *not changed.*

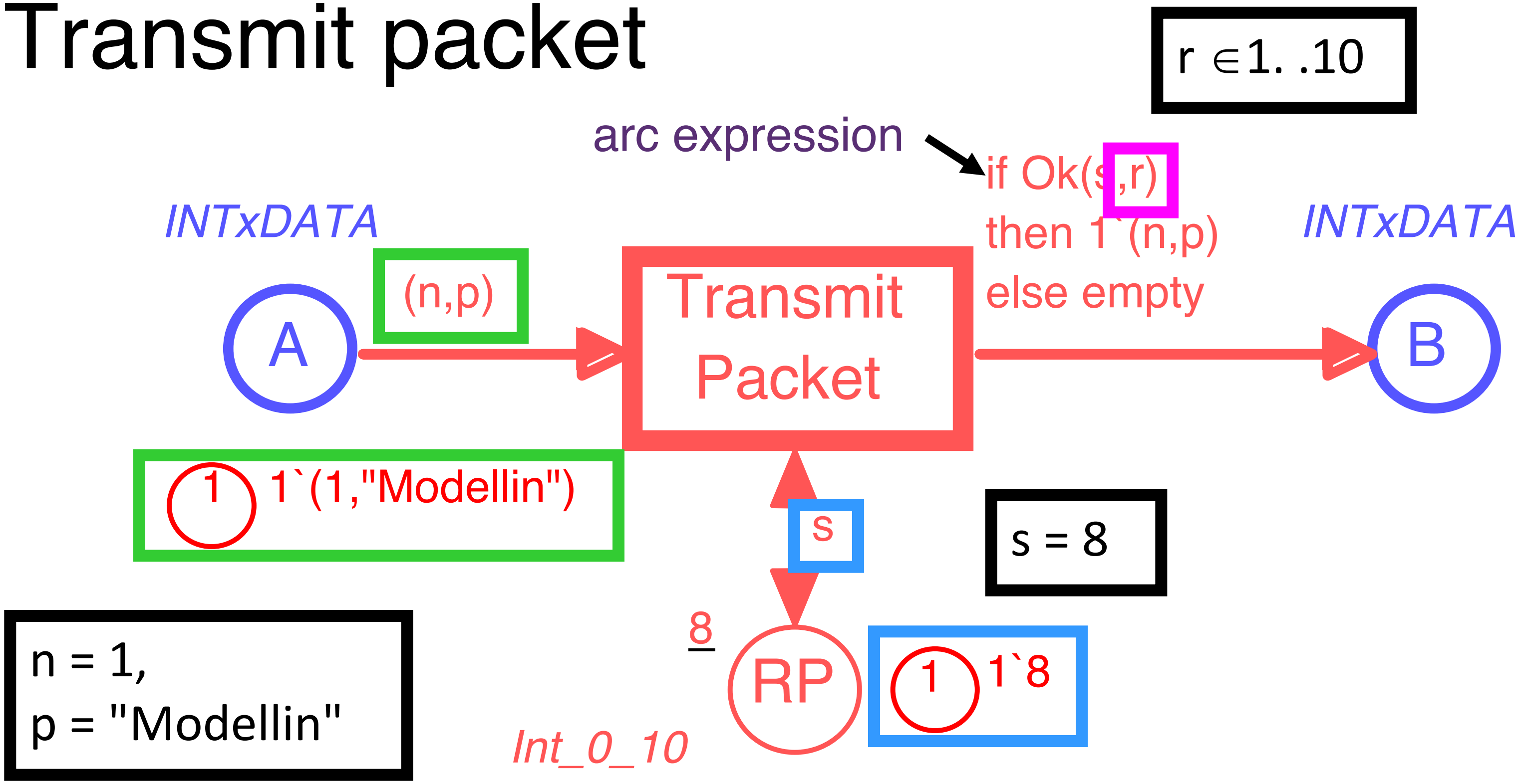


Simple protocol



UNIVERSITY OF WOLLONGONG AUSTRALIA

Transmit packet



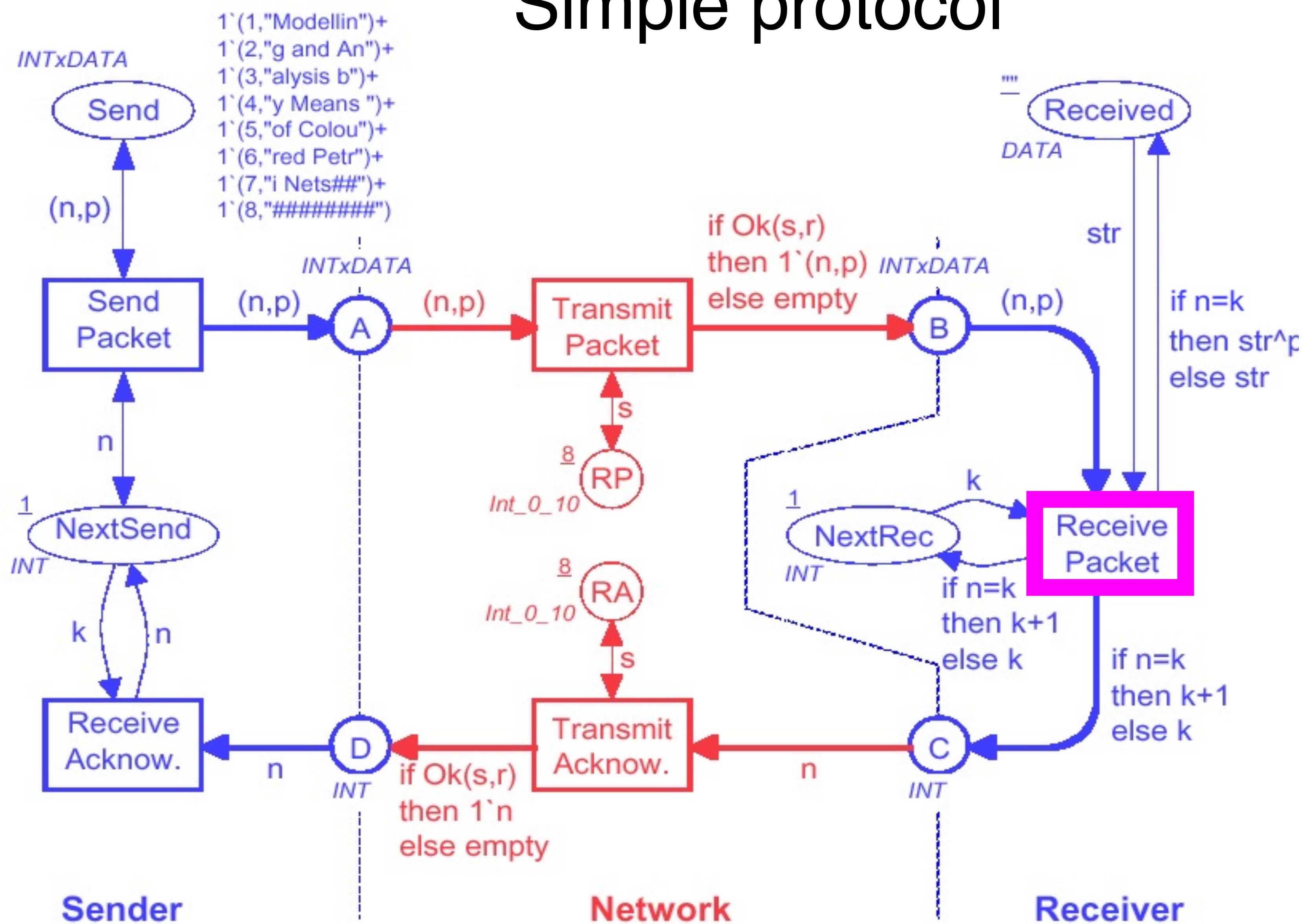
- All *enabled bindings* are on the form:
  - <n=1,p= "Modellin",s=8,r=...>
  - where r ∈1. .10

Loss of packets

```
if Ok(s,r)
then 1`(n,p)
else empty
```

- The *function* *Ok(s,r)* checks whether *$r \leq s$.*
  - *For r $\in$1. .8, Ok(s,r)=true.*
    The token is moved from A to B. This means that the packet is *successfully transmitted* over the network.
  - *For r $\in$ 9. .10, Ok(s,r)=false.*
    No token is added to B. This means that the packet is *lost.*
- The CPN simulator makes *random choices* between bindings: 80% chance for successful transfer.

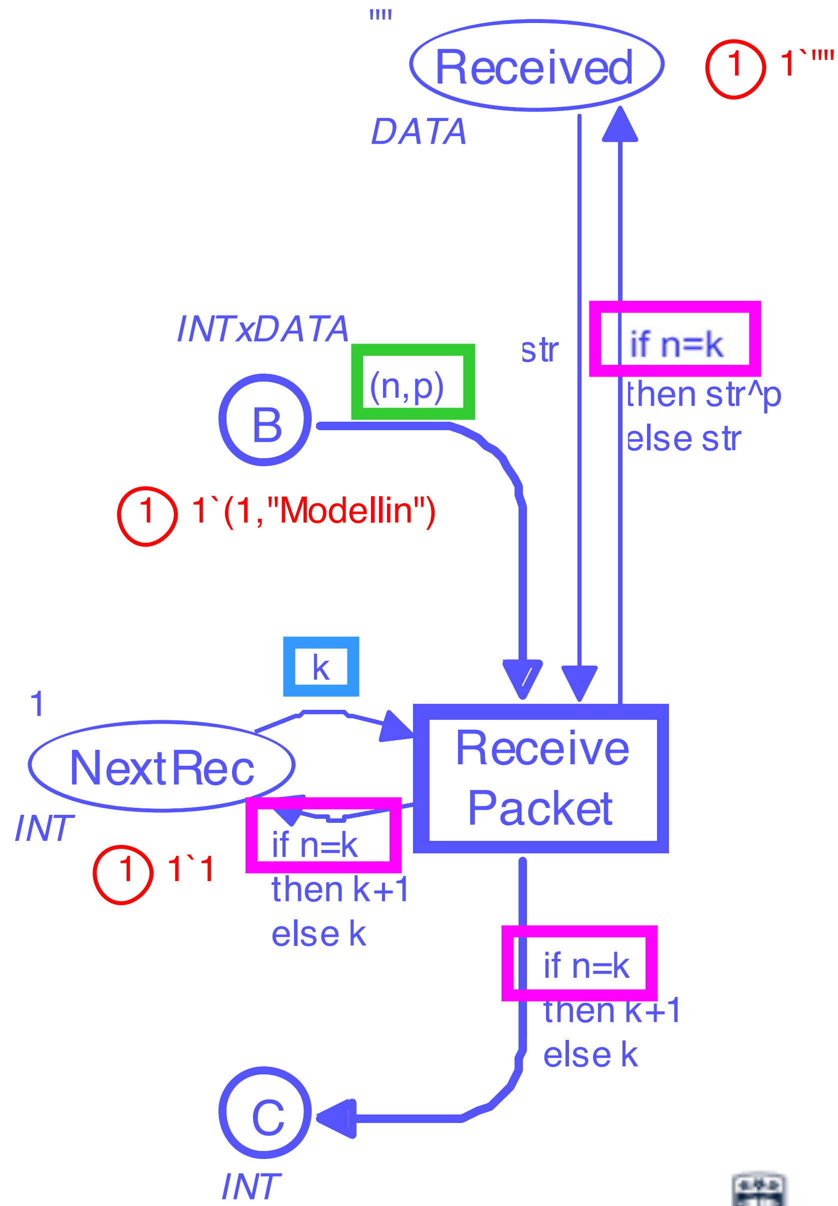
Simple protocol



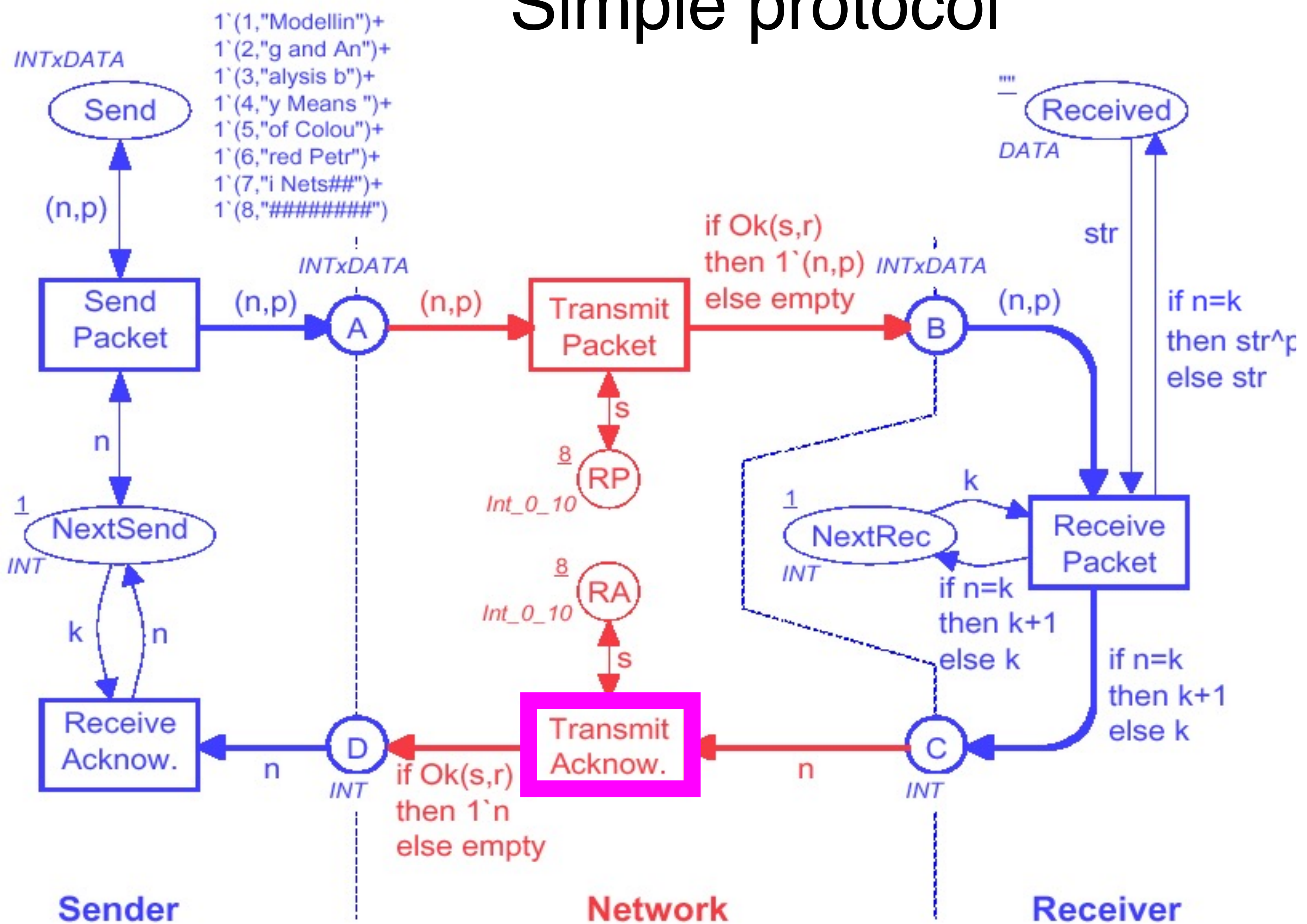
UNIVERSITY OF WOLLONGONG AUSTRALIA

Receive packet

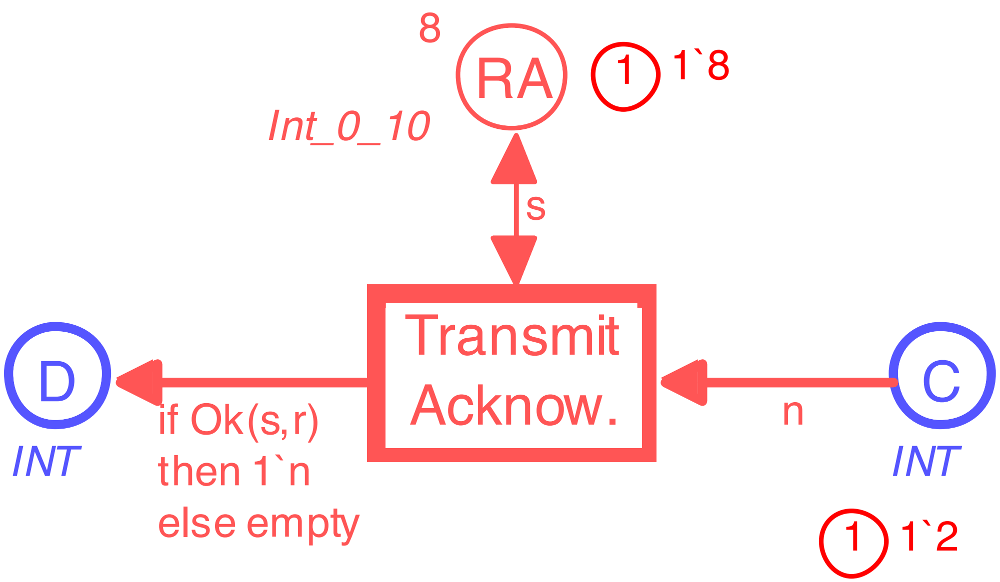
- The number of the *incoming packet* n and the number of the *expected packet* k are *compared.*



Simple protocol

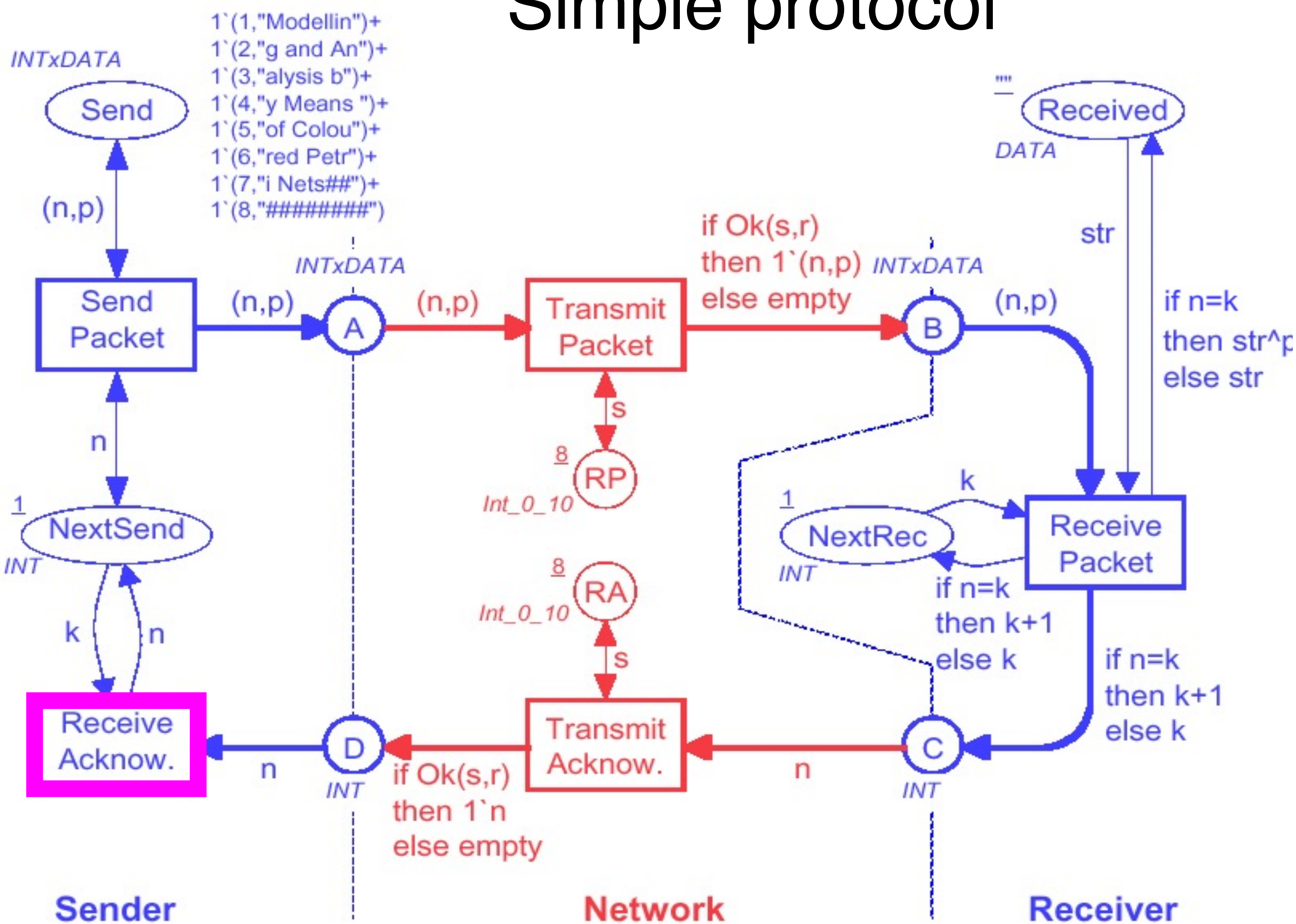


Transmit acknowledgement

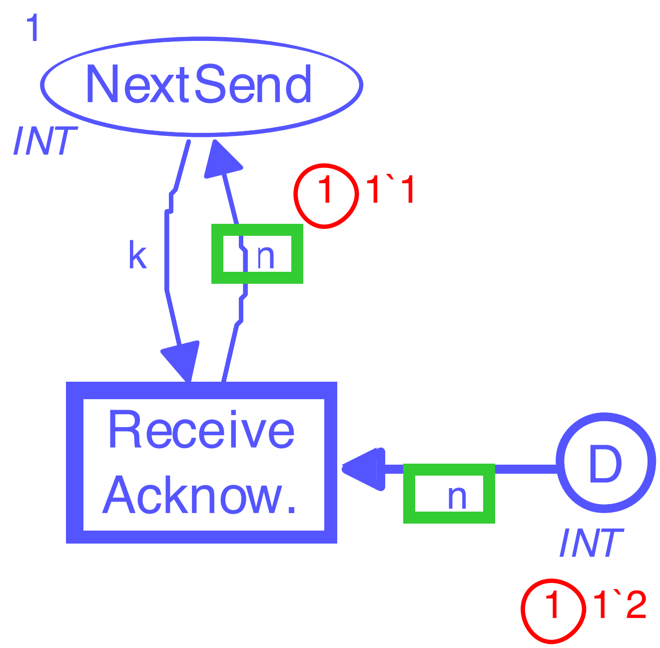


- This transition works in a similar way as *Transmit Packet.*
- The marking of *RA* determines the *success rate.*

Simple protocol

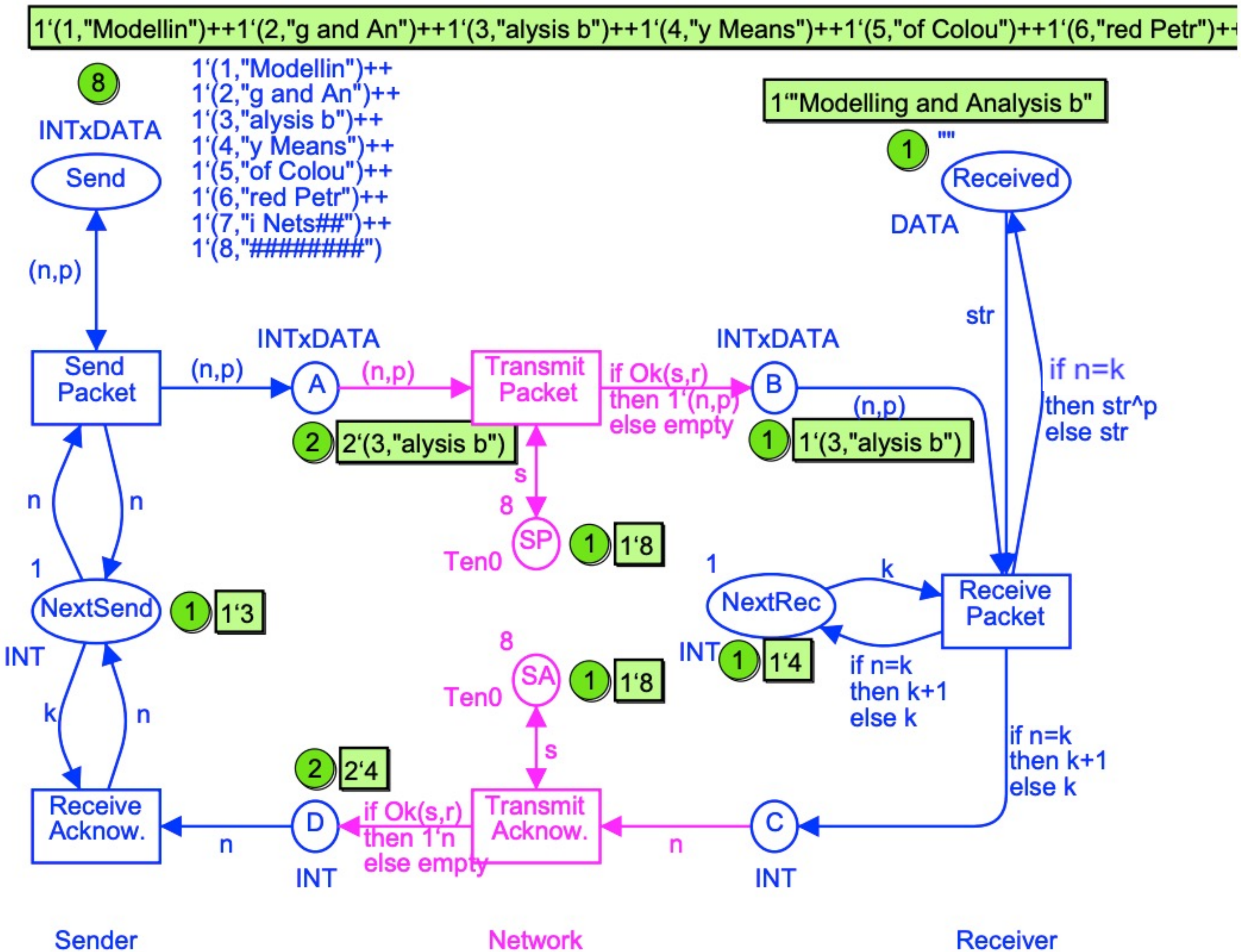


Receive acknowledgement

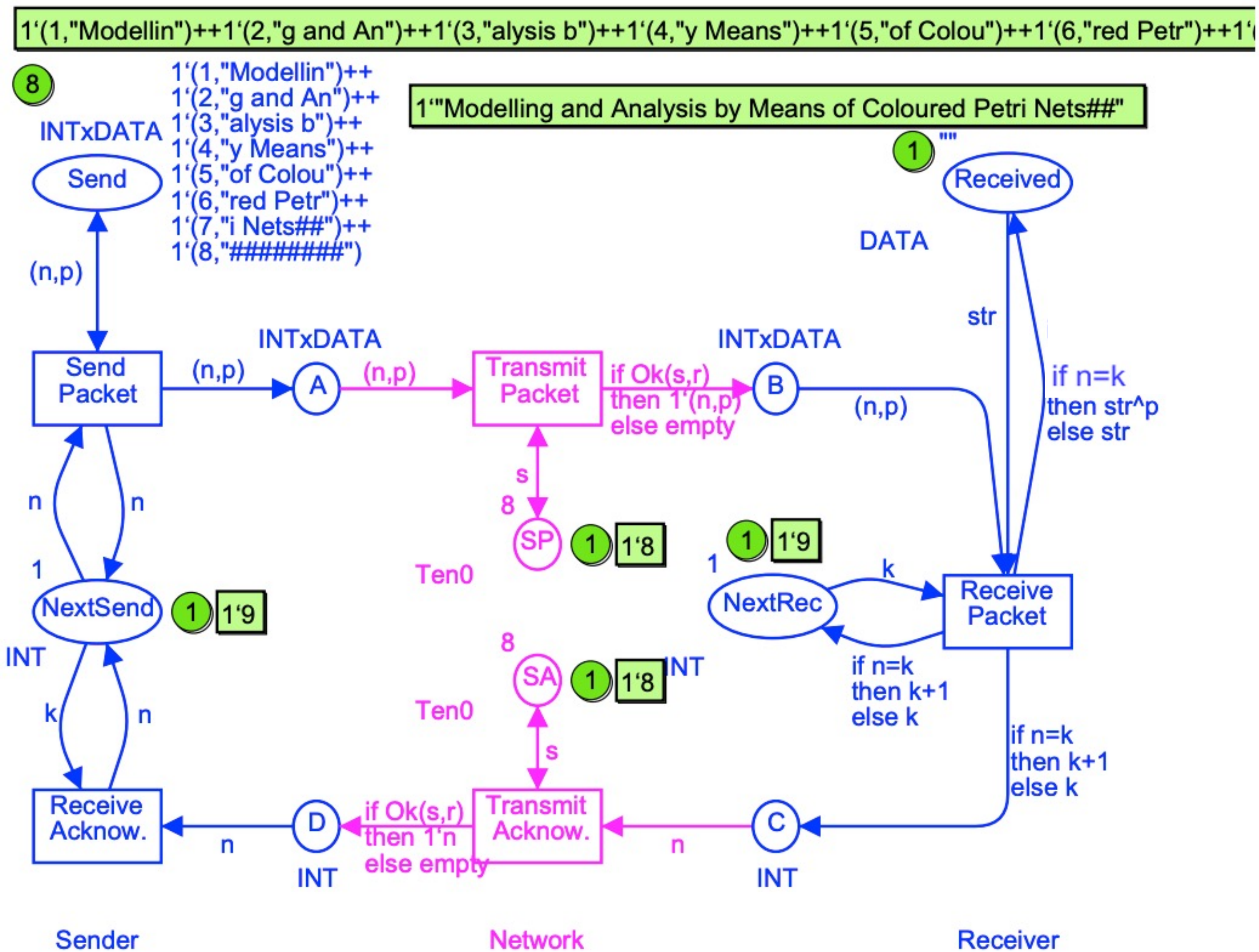


- When an acknowledgement arrives to the *Sender*, it is used to update the *NextSend* counter.
  - In this case the counter value becomes 2, and hence the *Sender* will begin to send *packet number 2.*

Intermediate Marking



Final Marking



Computer tools

Design/CPN was developed in the late 80'ies and early 90'ies.

- Today it is the *most widely used* Petri net package.
- *750 different organisations* in *50 countries*
- including *200 commercial companies.*

CPN Tools are the next generation of tool support for coloured Petri Nets.

- CPN Tools is expected to *replace Design/CPN* and obtain the same number of users.
- http://cpntools.org/
- http://cpntools.org/category/documentation/doc-examples/
- http://cpntools.org/2018/01/09/simple-protocol-example/
- https://www.youtube.com/watch?v=jO7RnCojIck

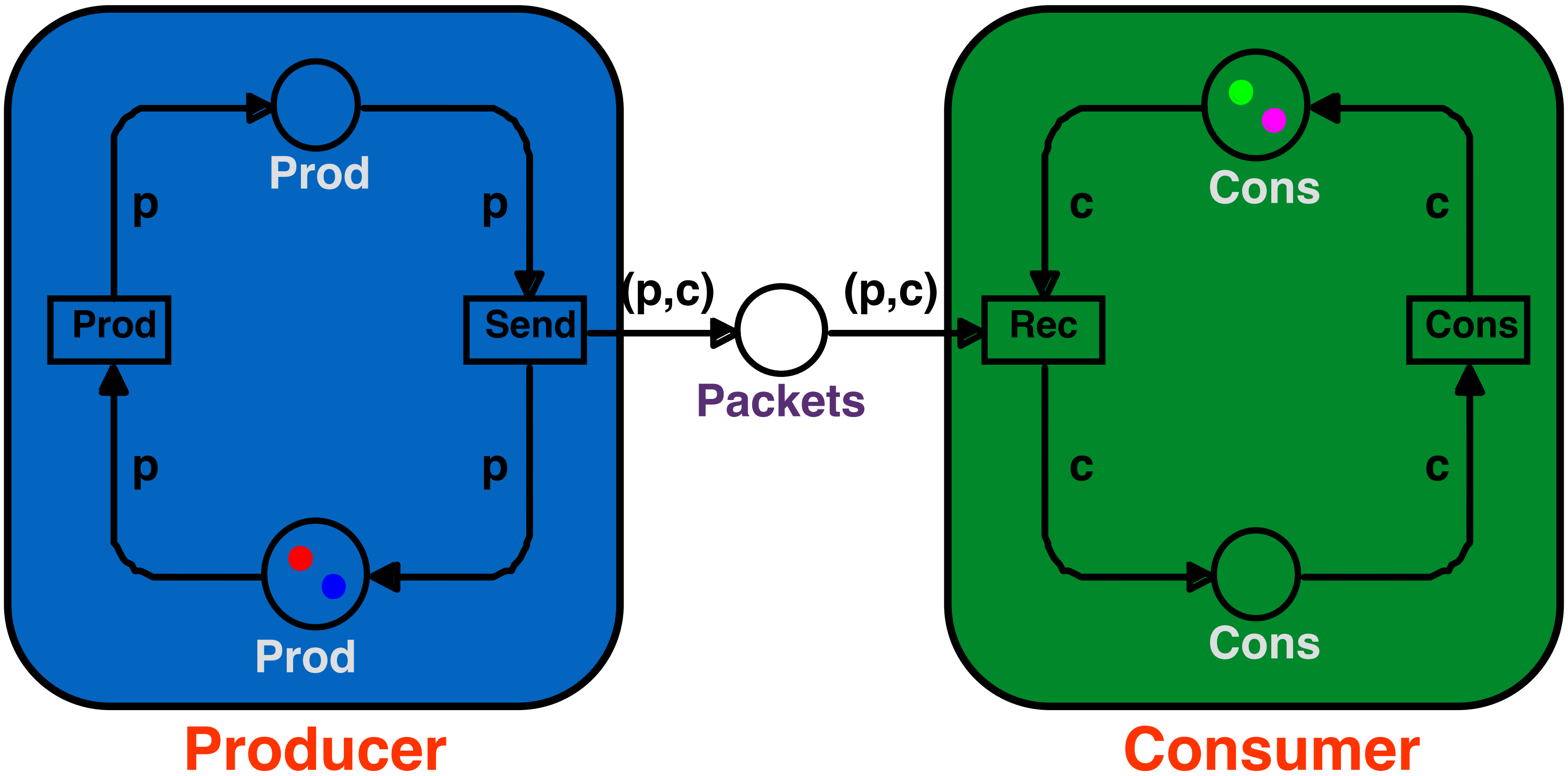
CP-nets are used for large systems

- A CPN model consists of a number of *modules*.
  - Also called *subnets* or *pages.*
  - Well-defined *interfaces.*
- A typical *industrial application* of CP-nets has:
  - 10-200 modules.
  - 50-1000 places and transitions.
  - 10-200 types.
- Industrial applications of this size would be *totally impossible* without:
  - Data types and token values.
  - Modules.
  - Tool support.

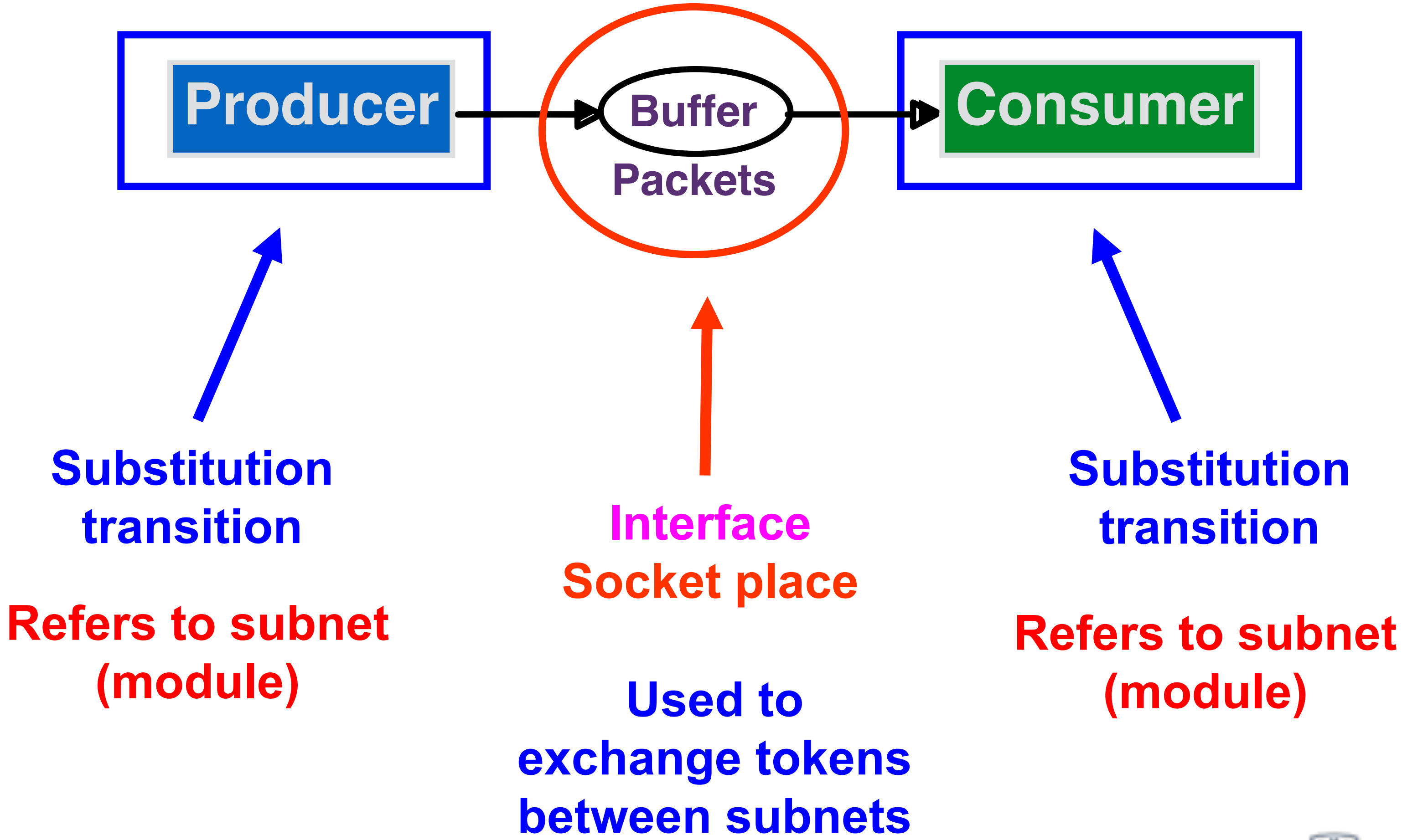
Hierarchical descriptions

- We use *modules* to *structure* *large* and *complex* descriptions.
- Modules allow us to *hide details* that we do not want to consider at a certain *level of abstraction*.
- Modules have *well-defined interfaces,* consisting of *socket* and *port places,* through which the modules *exchange tokens* with each other.
- Modules can be *reused.*

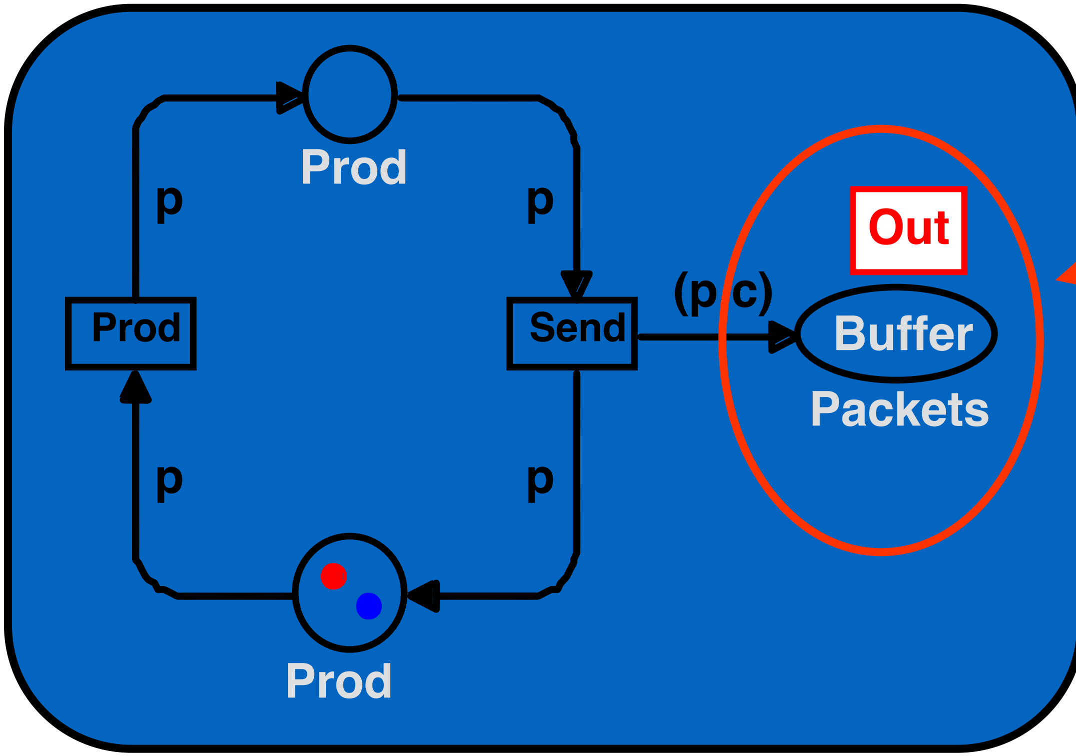
Hierarchical descriptions (modules)



Abstract view



Producer module



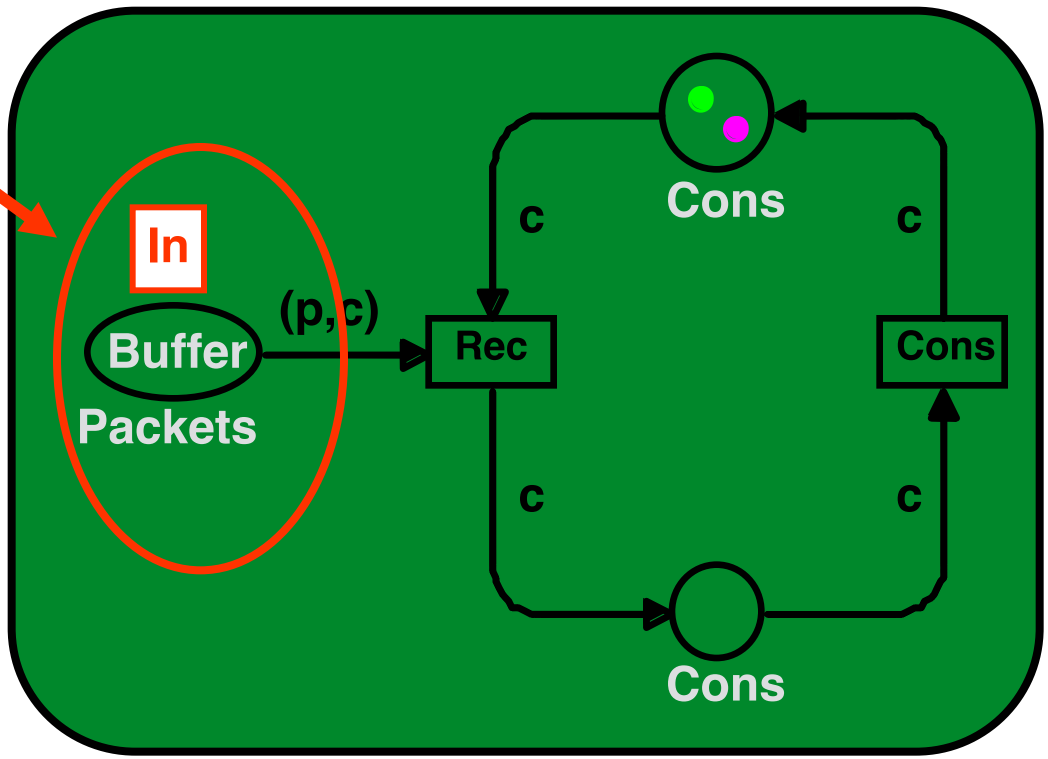
Interface
Output port

Used to export tokens to the rest of the net

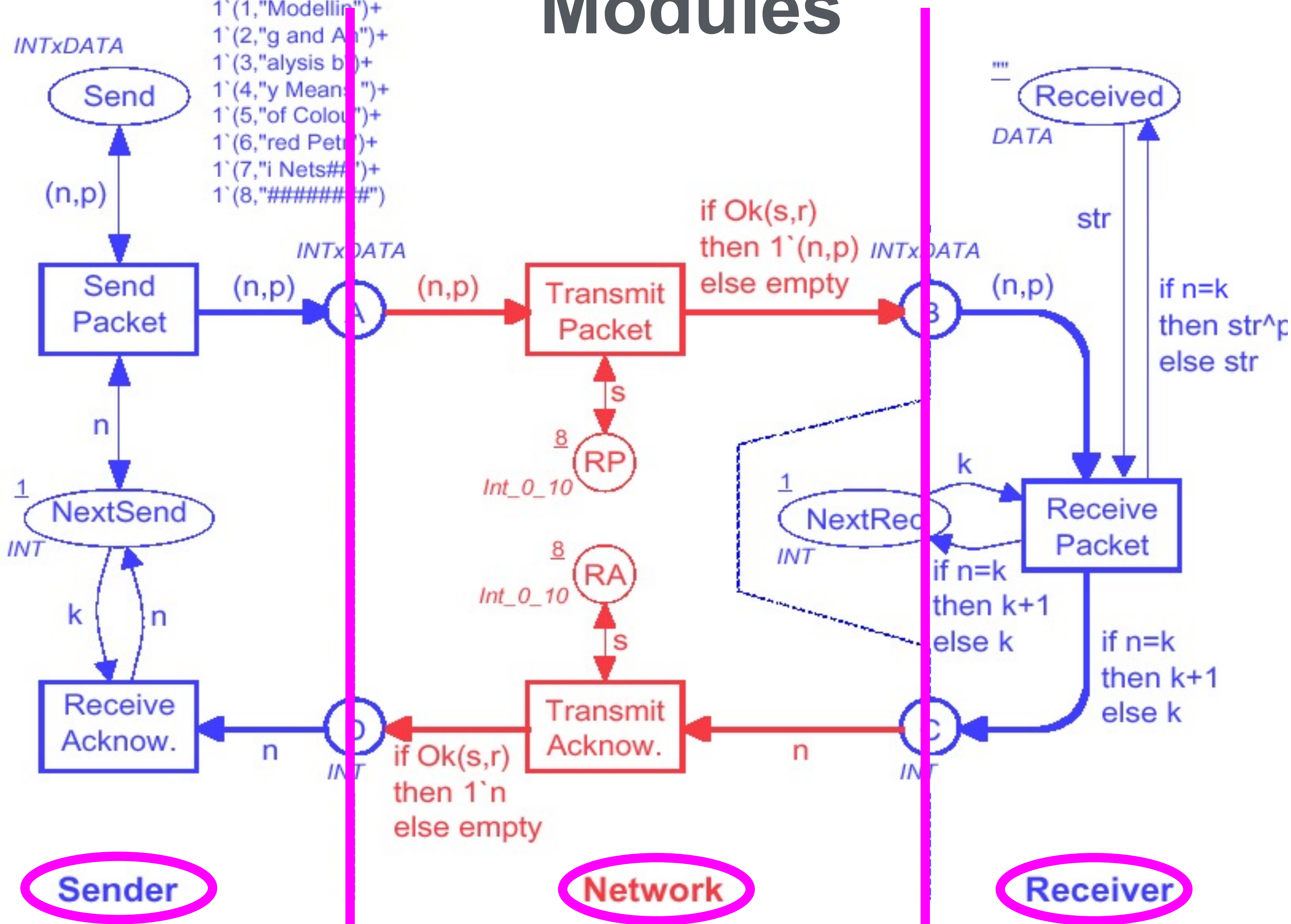
Consumer module

Interface
Input port

Used to
import tokens
from the rest of
the net



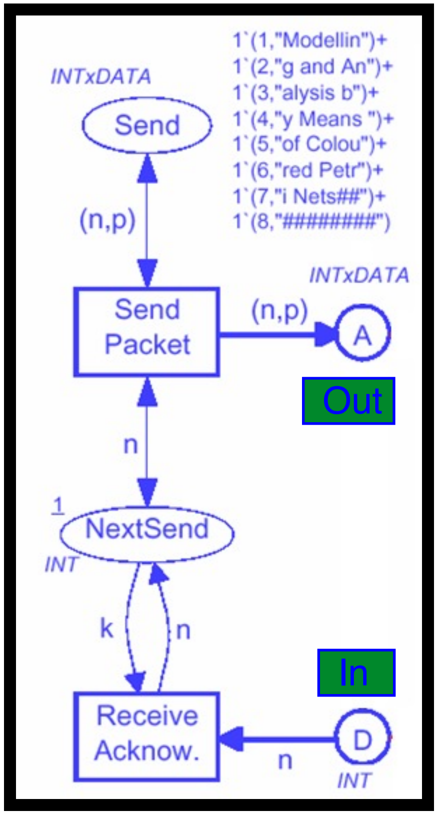
Modules



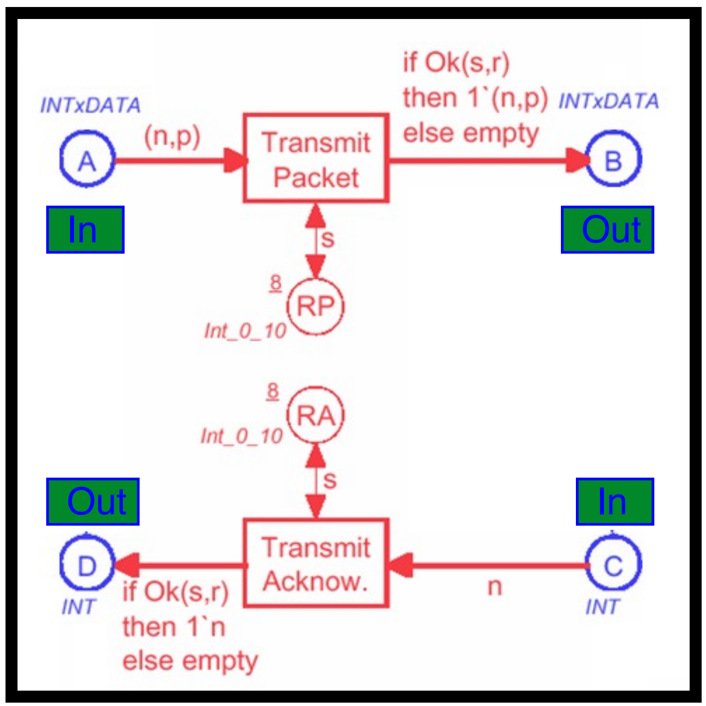
INTxDATA
Send
1`(1,"Modellin")+
1`(2,"g and An")+
1`(3,"alysis b")+
1`(4,"y Means ")+
1`(5,"of Colou")+
1`(6,"red Petr")+
1`(7,"i Nets##")+
1`(8,"########")
(n,p)
Send Packet
(n,p)
INTxDATA
A
(n,p)
Transmit Packet
if Ok(s,r) then 1`(n,p) else empty
INTxDATA
B
(n,p)
""
Received
DATA
str
if n=k then str^p else str
n
s
8
RP
Int_0_10
1
NextSend
INT
k
n
Receive Packet
k
1
NextRec
INT
if n=k then k+1 else k
8
RA
Int_0_10
s
if n=k then k+1 else k
Receive Acknow.
n
D
INT
if Ok(s,r) then 1`n else empty
Transmit Acknow.
n
C
INT
Sender
Network
Receiver

Three different modules

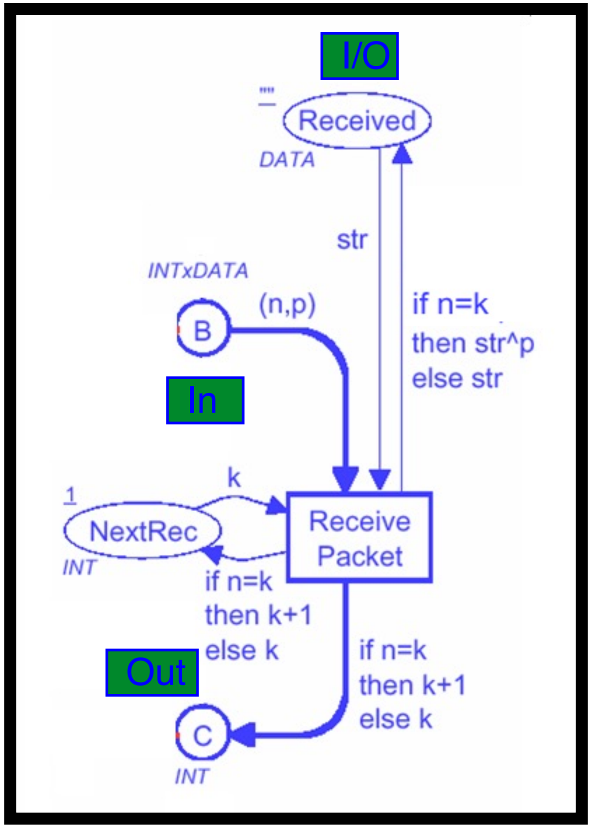
Sender



Network



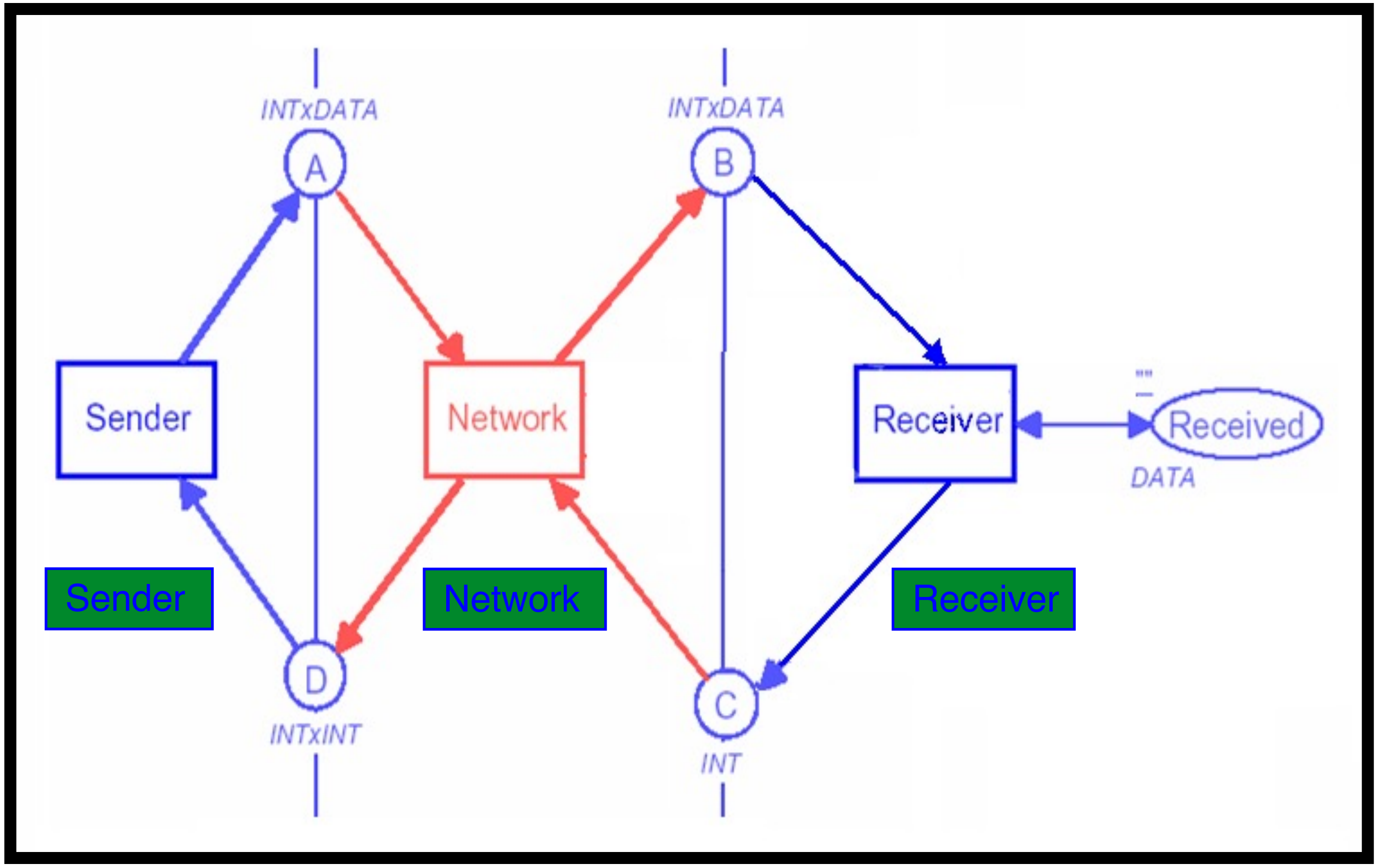
Receiver



Port places are used to *exchange tokens* between modules.

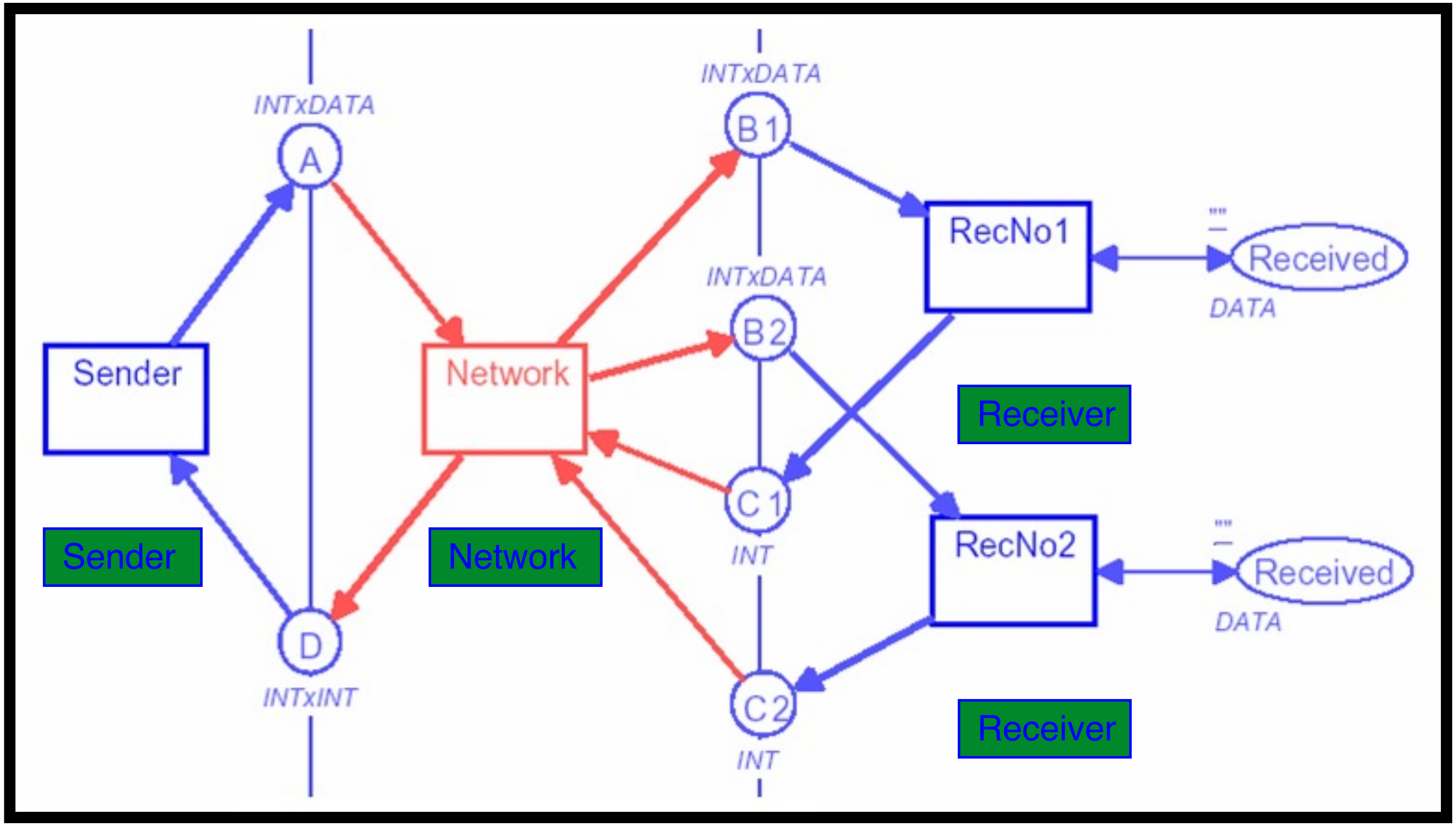
Abstract view

Protocol



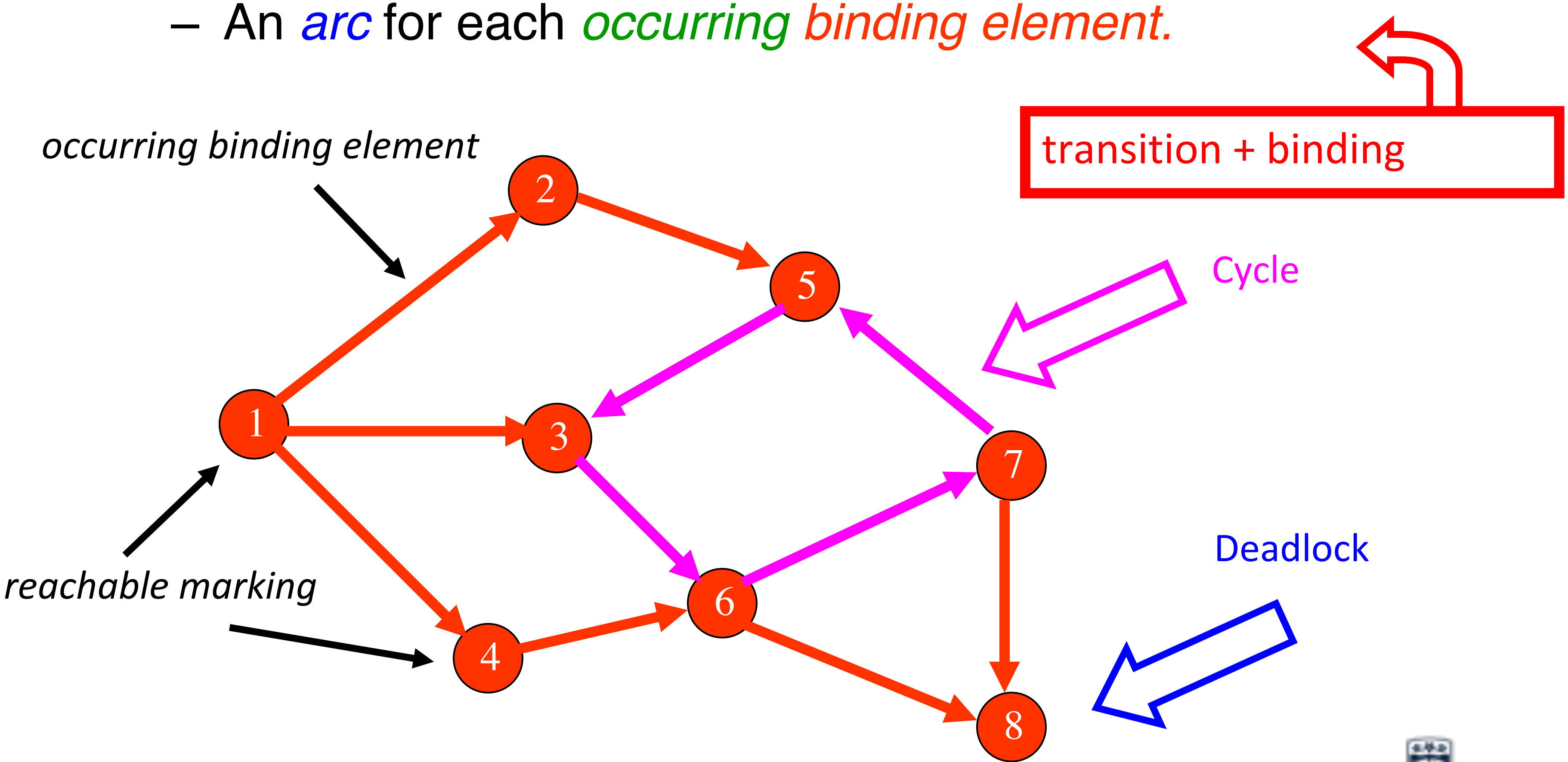
Modules can be reused

Protocol

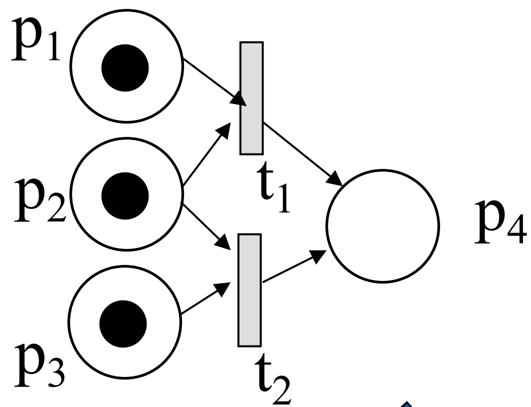


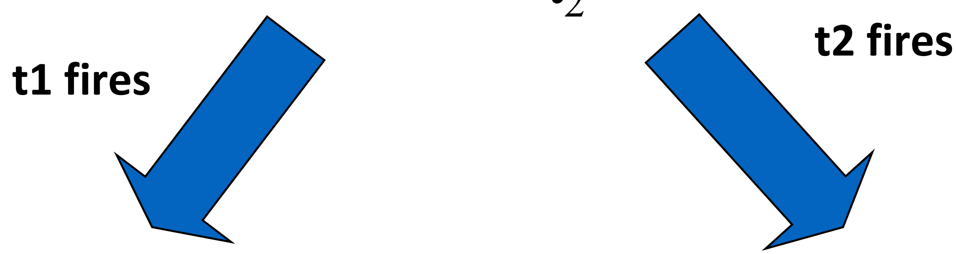
State spaces (For analysis)

- A *state space* is a *directed graph* with:
  - A *node* for each *reachable marking* (i.e., state).
  - An *arc* for each *occurring binding element.*

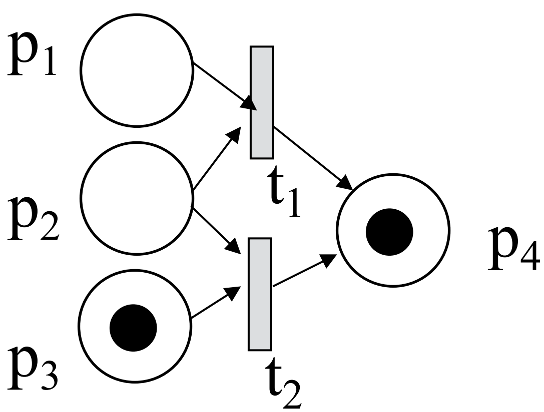


State 1 $\mu 0=(1,1,1,0)$

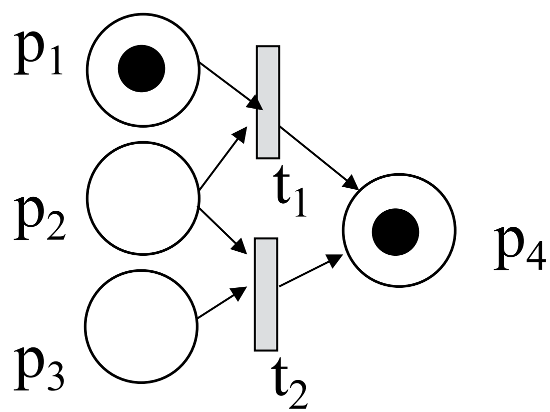




State 2 $\mu 1=(0,0,1,1)$



State 3 $\mu 2=(1,0,0,1)$



State space tool

- State spaces are often *very large.*
- The *CPN state space tool* allows the user to:
  - *Generate* state spaces.
  - *Analyse* state spaces to obtain information about the *behaviour* of the modelled system.=

State space report

- Generation of the *state space report* takes often only a *few seconds*.
  - The report contains a lot of useful information about the *behaviour* of the CP-net.
  - The report is excellent for *locating errors* or to *increase our confidence* in the *correctness* of the system.

State spaces - pros/cons

- State spaces are *powerful* and *easy* to use.
  - *Construction* and *analysis* can be *automated.*
  - *No need* to know the *mathematics* behind the analysis methods.

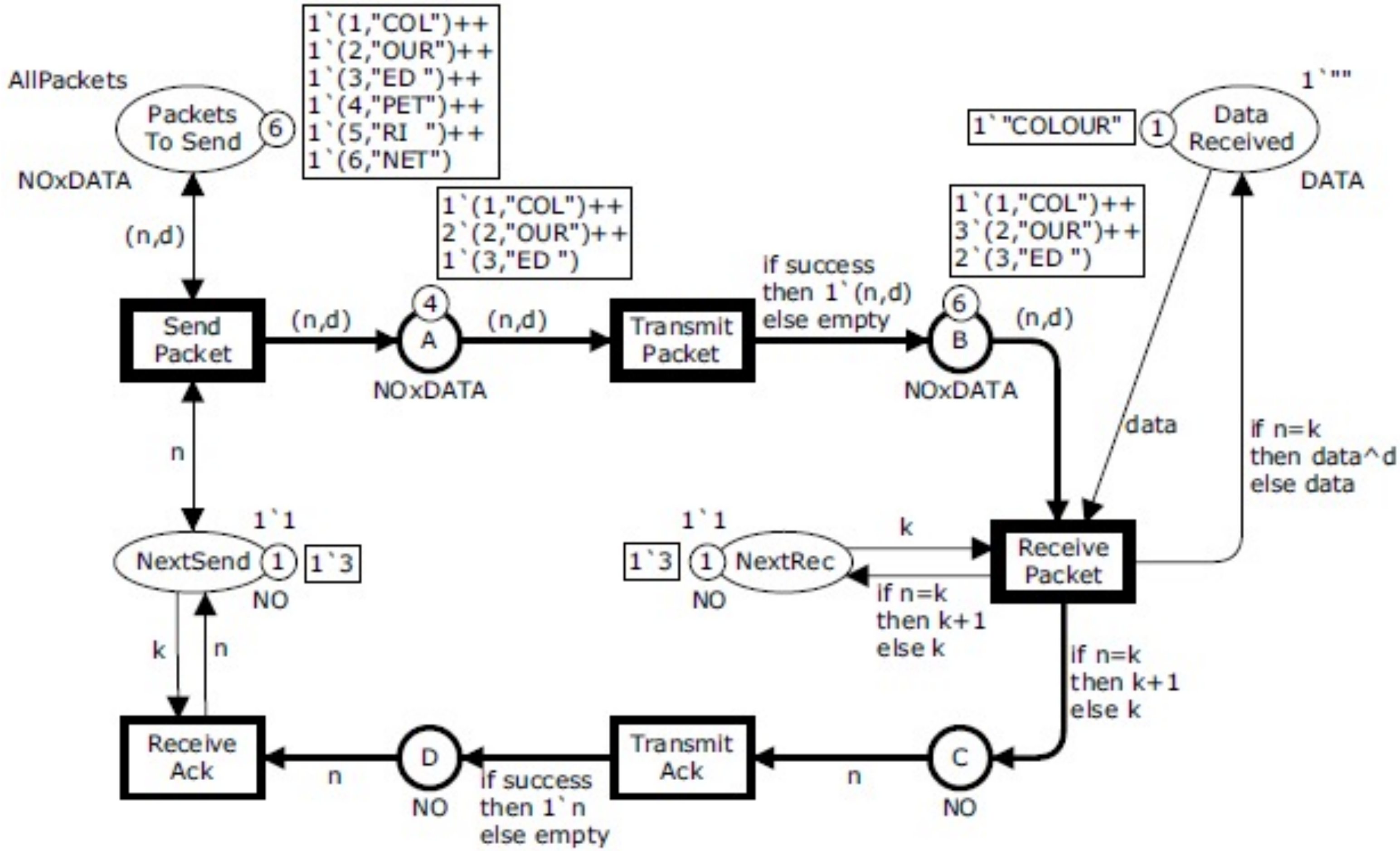
- The main drawback is the *state explosion*, i.e., the *size* of the state space.

CPN Formalisation

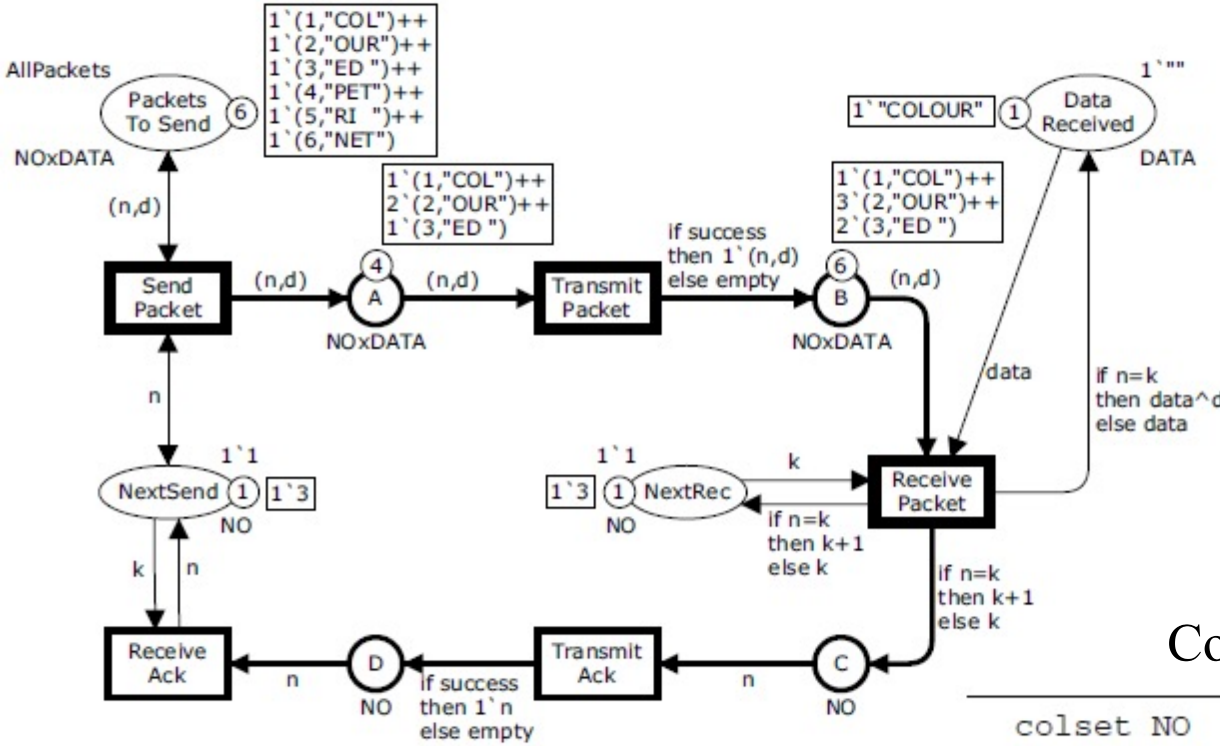
- A *CPN*, is a 9-tuple, *CPN*=$(P, T, A, \Sigma, V, C, G, E, I)$, where

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**.
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 $v \in V$.
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 $Type[I(p)] = C(p)_{MS}$.

CPN Formalisation Example



CPN Formalisation Example

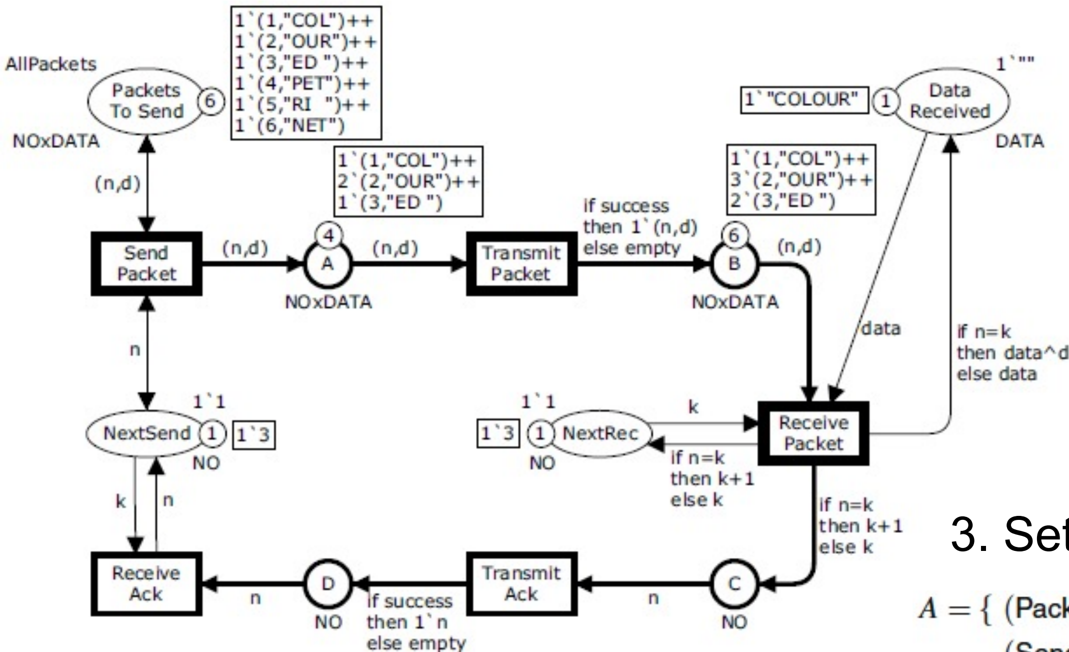


Colour sets and variables

```
colset NO      = int;
colset DATA    = string;
colset NOxDATA = product NO * DATA;
colset BOOL    = bool;

var n, k    : NO;
var d, data : DATA;
var success : BOOL;
```

CPN Formalisation Example



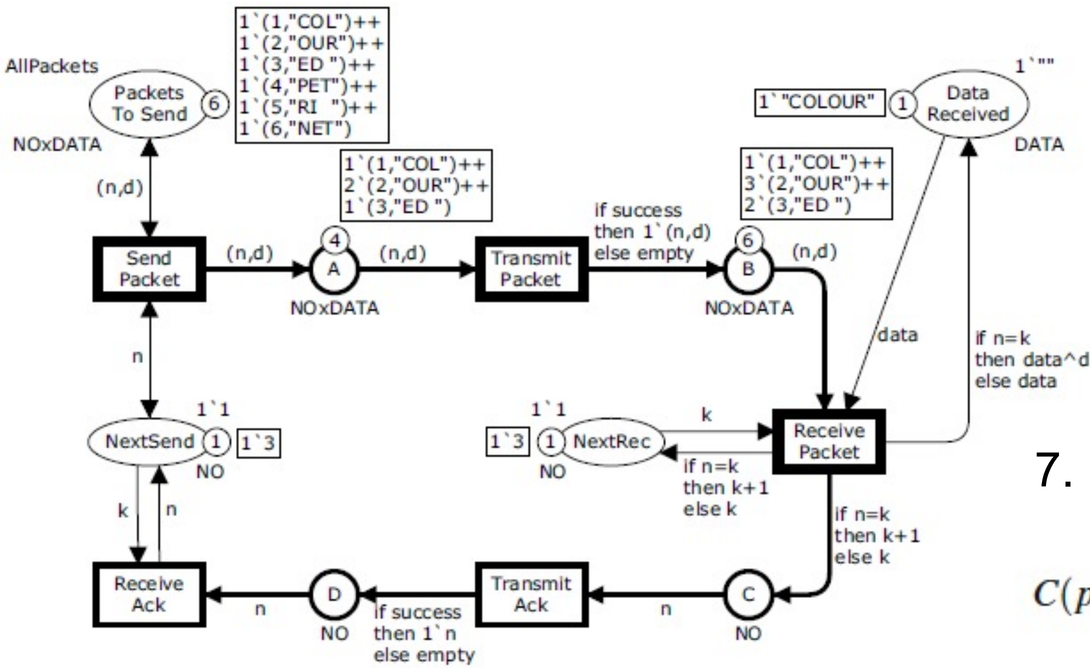
3. Set of arcs:

$$A = \{\ (\text{PacketsToSend}, \text{SendPacket}), (\text{SendPacket}, \text{PacketsToSend}),\\ (\text{SendPacket}, \text{A}), (\text{A}, \text{TransmitPacket}), (\text{TransmitPacket}, \text{B}),\\ (\text{B}, \text{ReceivePacket}), (\text{NextRec}, \text{ReceivePacket}), (\text{ReceivePacket}, \text{NextRec}),\\ (\text{DataReceived}, \text{ReceivePacket}), (\text{ReceivePacket}, \text{DataReceived}),\\ (\text{ReceivePacket}, \text{C}), (\text{C}, \text{TransmitAck}), (\text{TransmitAck}, \text{D}), (\text{D}, \text{ReceiveAck})\\ (\text{ReceiveAck}, \text{NextSend}), (\text{NextSend}, \text{ReceiveAck}),\\ (\text{NextSend}, \text{SendPacket}), (\text{SendPacket}, \text{NextSend})\ \}$$

1. Set of places: $P = \{\ \text{PacketsToSend}, \text{A}, \text{B}, \text{DataReceived}, \text{NextRec}, \text{C}, \text{D}, \text{NextSend}\ \}$

2. Set of transitions: $T = \{\ \text{SendPacket}, \text{TransmitPacket}, \text{ReceivePacket}, \text{TransmitAck}, \text{ReceiveAck}\ \}$

CPN Formalisation Example



7. Set of colour set functions

$$C(p) = \begin{cases} \mathrm{NO} & \text{if } p \in \{\mathsf{NextSend}, \mathsf{NextRec}, \mathsf{C}, \mathsf{D}\} \\ \mathrm{DATA} & \text{if } p = \mathsf{DataReceived} \\ \mathrm{NOxDATA} & \text{if } p \in \{\mathsf{PacketsToSend}, \mathsf{A}, \mathsf{B}\} \end{cases}$$

4. Set of colour sets: $\Sigma = \{\,\mathrm{NO}, \mathrm{DATA}, \mathrm{NOxDATA}, \mathrm{BOOL}\,\}$

5. Set of variables:

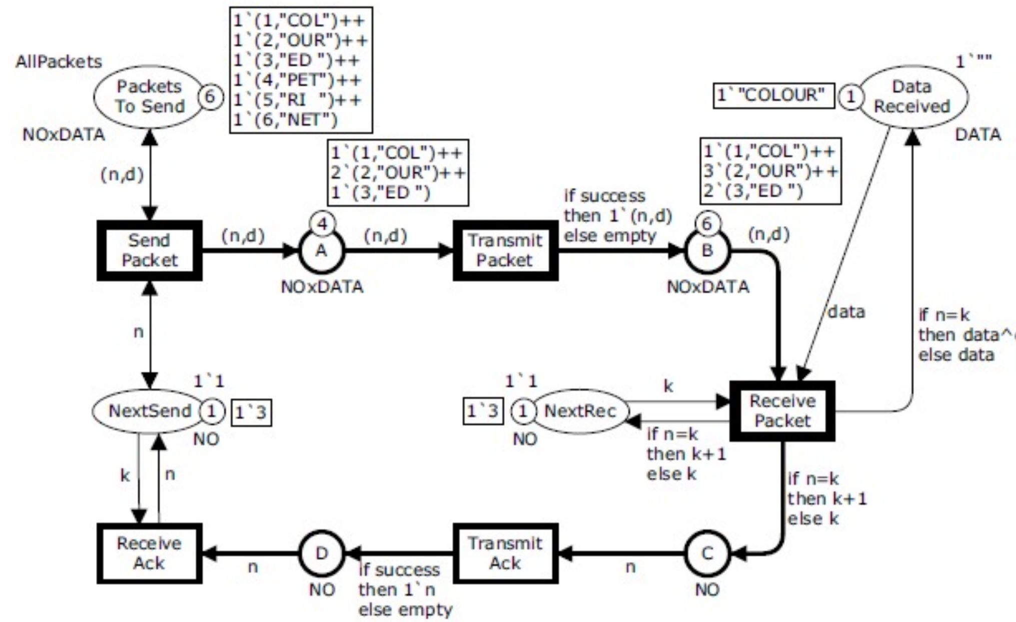
$$V = \{\,\mathtt{n}:\mathrm{NO}, \mathtt{k}:\mathrm{NO}, \mathtt{d}:\mathrm{DATA}, \mathtt{data}:\mathrm{DATA}, \mathtt{success}:\mathrm{BOOL}\,\}$$

6. Set of guard functions: $G(t) = \mathtt{true} \text{ for all } t \in T$

CPN Formalisation Example

8. Set of arc expression functions

$$E(a) = \begin{cases} \texttt{1`(n,d)} & \text{if } a \in \{(\mathrm{PacketsToSend}, \mathrm{SendPacket}), \\ & \quad (\mathrm{SendPacket}, \mathrm{PacketsToSend}), \\ & \quad (\mathrm{SendPacket}, \mathrm{A}), \\ & \quad (\mathrm{A}, \mathrm{TransmitPacket}), \\ & \quad (\mathrm{B}, \mathrm{ReceivePacket})\} \\ \texttt{if success then 1`(n,d) else empty} & \text{if } a = (\mathrm{TransmitPacket}, \mathrm{B}) \\ \texttt{1`k} & \text{if } a \in \{(\mathrm{NextRec}, \mathrm{ReceivePacket}), \\ & \quad (\mathrm{NextSend}, \mathrm{ReceiveAck})\} \\ \texttt{1`(if n=k then k+1 else k)} & \text{if } a \in \{(\mathrm{ReceivePacket}, \mathrm{NextRec}), \\ & \quad (\mathrm{ReceivePacket}, \mathrm{C})\} \\ \texttt{1`data} & \text{if } a = (\mathrm{DataReceived}, \mathrm{ReceivePacket}) \\ \texttt{1`(if n=k then data\^{}d else data)} & \text{if } a = (\mathrm{ReceivePacket}, \mathrm{DataReceived}) \\ \texttt{1`n} & \text{if } a \in \{(\mathrm{C}, \mathrm{TransmitAck}), \\ & \quad (\mathrm{D}, \mathrm{ReceiveAck}), \\ & \quad (\mathrm{ReceiveAck}, \mathrm{NextSend}), \\ & \quad (\mathrm{NextSend}, \mathrm{SendPacket}), \\ & \quad (\mathrm{SendPacket}, \mathrm{NextSend})\} \\ \texttt{if success then 1`n else empty} & \text{if } a = (\mathrm{TransmitAck}, \mathrm{D}) \end{cases}$$

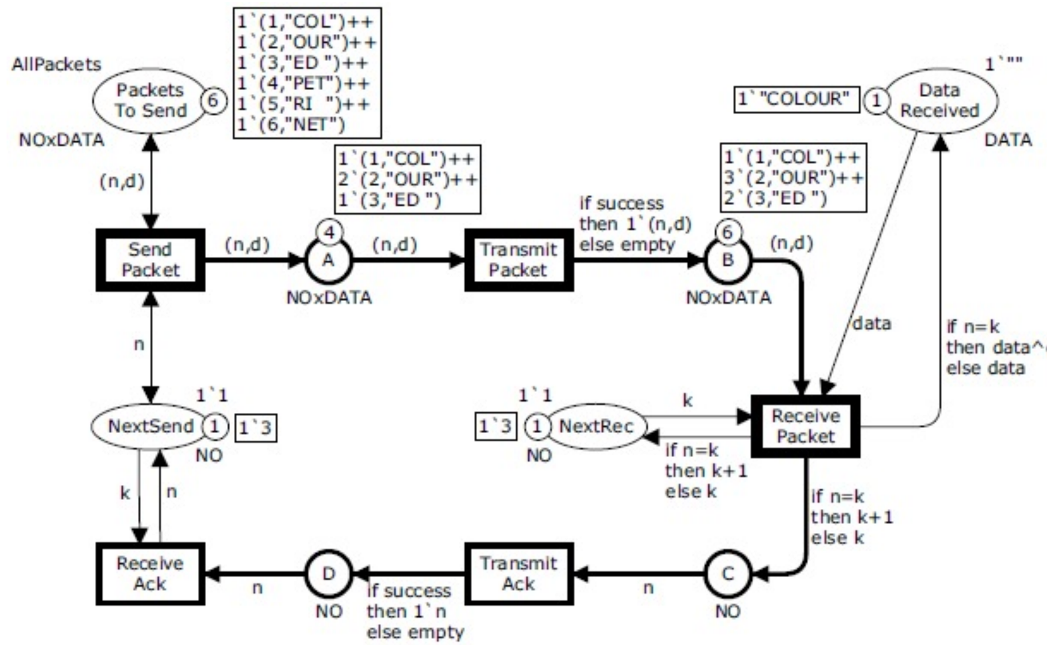


9. Set of initialisation functions

$$I(p) = \begin{cases} \texttt{AllPackets} & \text{if } p = \mathrm{PacketsToSend} \\ \texttt{1`1} & \text{if } p \in \{\mathrm{NextSend}, \mathrm{NextRec}\} \\ \texttt{1`""} & \text{if } p = \mathrm{DataReceived} \\ \boxed{\emptyset_{MS}} & \text{otherwise} \end{cases}$$

the empty multiset over *S*

CPN Formalisation Example



Marking of a CPN:

A marking ***M*** is a function that maps each place ***p*** into a multiset of values ***M(p)*** representing the marking of ***p***, i.e., $M(p) \in C(p)_{MS}$.

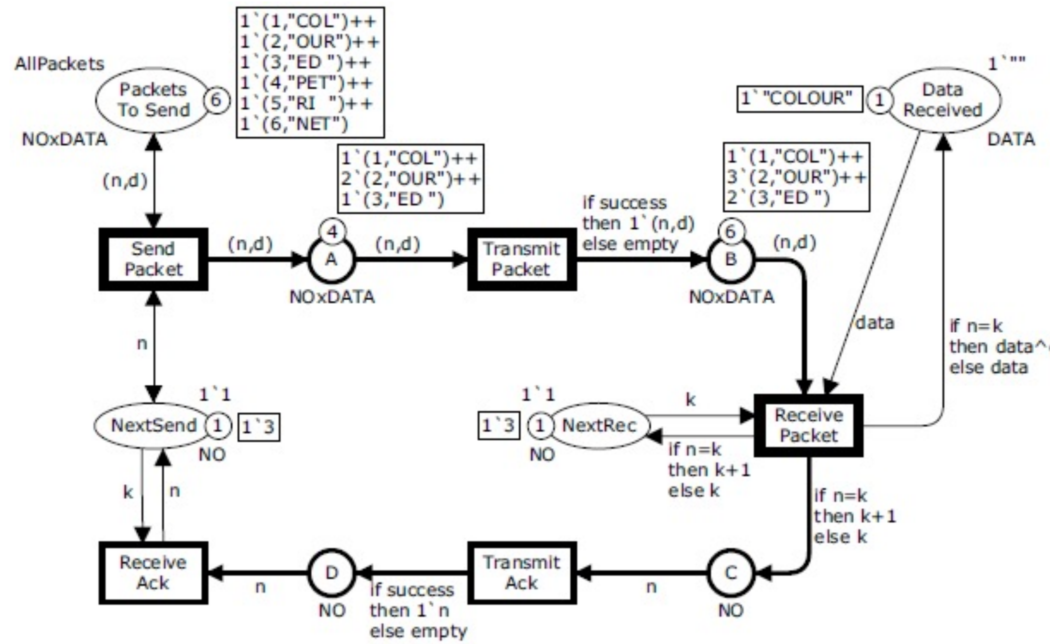
Example:

$$M(p) = \begin{cases} \texttt{1`(1,"COL") ++ 1`(2,"OUR") ++} & \\ \texttt{1`(3,"ED ") ++ 1`(4,"PET") ++} & \text{if } p = \text{PacketsToSend} \\ \texttt{1`(5,"RI ") ++ 1`(6,"NET")} & \\ & \\ \texttt{1`3} & \text{if } p \in \{\text{NextSend}, \text{NextRec}\} \\ & \\ \texttt{1`"COLOUR"} & \text{if } p = \text{DataReceived} \\ & \\ \texttt{1`(1,"COL") ++ 2`(2,"OUR") ++} & \text{if } p = \text{A} \\ \texttt{1`(3,"ED ")} & \\ & \\ \texttt{1`(1,"COL") ++ 3`(2,"OUR") ++} & \text{if } p = \text{B} \\ \texttt{2`(3,"ED ")} & \\ & \\ \emptyset_{MS} & \text{if } p \in \{\text{C}, \text{D}\} \end{cases}$$

CPN Formalisation Example

Initial marking of a CPN:

A CPN has a distinguished initial marking, denoted as M_0 , obtained by evaluating the initialisation expressions.



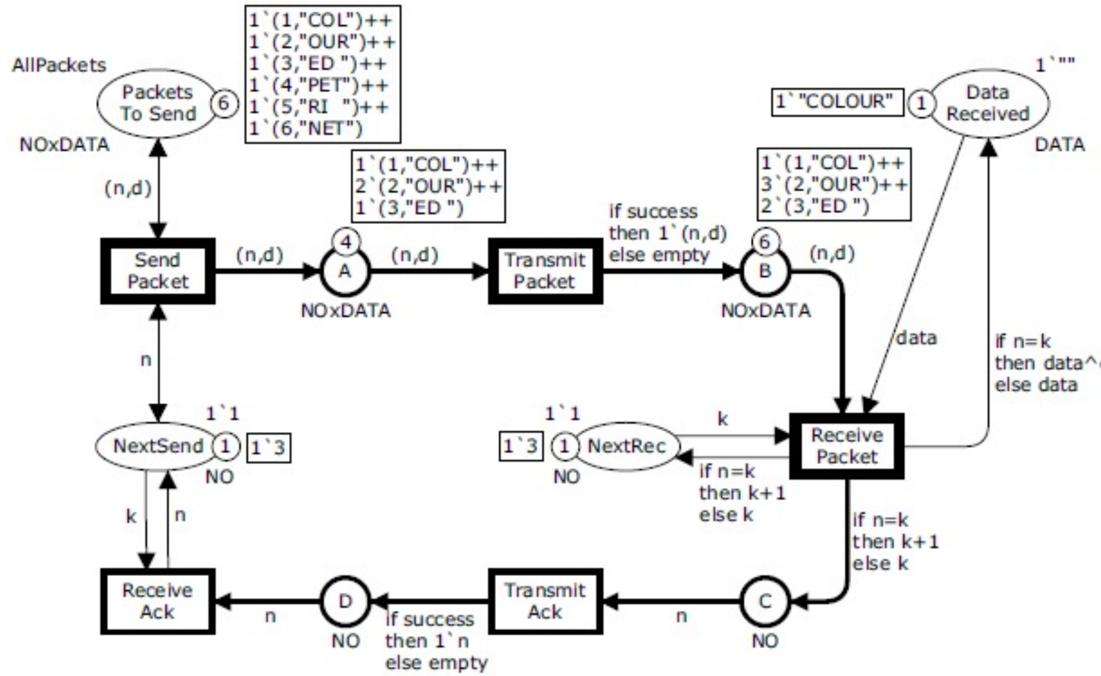
Example:

$$
M_0(p) = \begin{cases}
\texttt{1`(1,"COL") ++ 1`(2,"OUR") ++} & \\
\texttt{1`(3,"ED ") ++ 1`(4,"PET") ++} & \text{if } p = \text{PacketsToSend} \\
\texttt{1`(5,"RI ") ++ 1`(6,"NET")} & \\
\texttt{1`1} & \text{if } p \in \{\text{NextSend}, \text{NextRec}\} \\
\texttt{1`""} & \text{if } p = \text{DataReceived} \\
\emptyset_{MS} & \text{otherwise}
\end{cases}
$$

CPN Formalisation Example

Multisets:

To illustrate the definition of multisets, we use three multisets $\boldsymbol{m_p}$, $\boldsymbol{m_A}$, $\boldsymbol{m_B}$ over the colour set **NO×DATA** corresponding to the *markings of **PacketsToSend, A** and **B***



Example:

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

CPN Formalization Example

Coefficient:

- ***m(s)*** : the coefficient of ***s*** in ***m***, also the number of appearances of the element ***s*** in the multiset ***m***.

Example:
Consider the multiset $\boldsymbol{m_B}$ over the colour set **NO×DATA.**
The multiset $\boldsymbol{m_B}$ can be specified as the following function:

$$m_B(s) = \begin{cases} 1 & \text{if } s = (1, \texttt{"COL"}) \\ 3 & \text{if } s = (2, \texttt{"OUR"}) \\ 2 & \text{if } s = (3, \texttt{"ED "}) \\ 0 & \text{otherwise} \end{cases}$$

Assume that ***m*** is a multiset over a set $S = \{s_1, s_2, s_3, \dots\}$, then, ***m*** can be rewritten as:

$$\boxed{++}\sum_{s \in S} m(s)`s = m(s_1)`s_1 ++ m(s_2)`s_2 ++ m(s_3)`s_3 ++ \dots$$

Sum of multi-sets

CPN Formalization Example

Multiset Operations:

- **Addition**

Addition $m_1 + +m_2$ of two multisets m_1 and m_2 is obtained by adding the number of appearances $m_1(s)$ and the number of appearances $m_2(s)$,, i.e., $(\boldsymbol{m_1} + + \boldsymbol{m_2})(\boldsymbol{s}) = \boldsymbol{m_1}(\boldsymbol{s}) + \boldsymbol{m_2}(\boldsymbol{s})$.

Example:

$\boldsymbol{m_A} + +\boldsymbol{m_B}$ can be specified as the following function:

$$(m_A +\!+\, m_B)(s) = \begin{cases} 2 & \text{if } s = (1, \texttt{"COL"}) \\ 5 & \text{if } s = (2, \texttt{"OUR"}) \\ 3 & \text{if } s = (3, \texttt{"ED "}) \\ 0 & \text{otherwise} \end{cases}$$

CPN Formalization Example

- **Comparison**

A multiset m_1 is **smaller than or equal to** a multiset m_2, written as $\boldsymbol{m_1} \ll= \boldsymbol{m_2}$, if $m_1(s) \leq m_2(s)$.

Example: $\boldsymbol{m_A} \ll= \boldsymbol{m_B}$ ✓

$\boxed{\boldsymbol{m_A} \ll= \boldsymbol{m_p} \quad \times}$ ← The element 1`(2, "OUR") appears twice in $\boldsymbol{m_A}$, but only once in $\boldsymbol{m_p}$.

- **Subtraction**

Subtraction $m_2 - -m_1$ is obtained by subtracting the number of appearances $m_1(s)$ from the number of appearances $m_2(s)$ only when $m_2 \ll= m_1$, i.e., $\boldsymbol{(m_2 - - m_1)(s) = m_2(s) - m_1(s)}$.

Example:

$$(m_B -\!- m_A)(s) = \begin{cases} 1 & \text{if } s = (2,\texttt{"OUR"}) \\ 1 & \text{if } s = (3,\texttt{"ED "}) \\ 0 & \text{otherwise} \end{cases}$$

UNIVERSITY OF WOLLONGONG AUSTRALIA

CPN Formalization Example

- **Scalar Multiplication**

A multiset m is **multiplied by a scalar** $n \in \mathbb{N}$, written as $\boldsymbol{n} ** \boldsymbol{m}$, by multiplying the number of appearances $m(s)$ of each elements s by n, i.e., $(\boldsymbol{n} ** \boldsymbol{m})(\boldsymbol{s}) = \boldsymbol{n} * \boldsymbol{m}(\boldsymbol{s})$.

Example:

$$(4 ** m_B)(s) = \begin{cases} 4 & \text{if } s = (1, \texttt{"COL"}) \\ 12 & \text{if } s = (2, \texttt{"OUR"}) \\ 8 & \text{if } s = (3, \texttt{"ED "}) \\ 0 & \text{otherwise} \end{cases}$$

Conclusion

TOOLS
- editing
- simulation
- verification

THEORY
- models
- basic concepts
- analysis methods

PRACTICAL USE
- specification
- validation
- verification
- implementation

- One of the *reasons* for the *success* of CP-nets is the fact that we *simultaneously* have worked in *all three areas.*