Coloured Petri Nets

© Kurt Jensen
Department of Computer Science
University of Aarhus, Denmark

kjensen@daimi.au.dk
www.daimi.au.dk/~kjensen/

TOOLS
- **editing**
- **simulation**
- **verification**

THEORY
- **models**
- **basic concepts**
- **analysis methods**

PRACTICAL USE
- **specification**
- **validation**
- **verification**
- **implementation**

This slide set can be downloaded from:
http://www.daimi.au.dk/CPnets/slides/

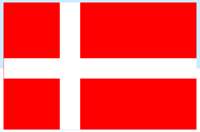
What is a Coloured Petri Net?

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

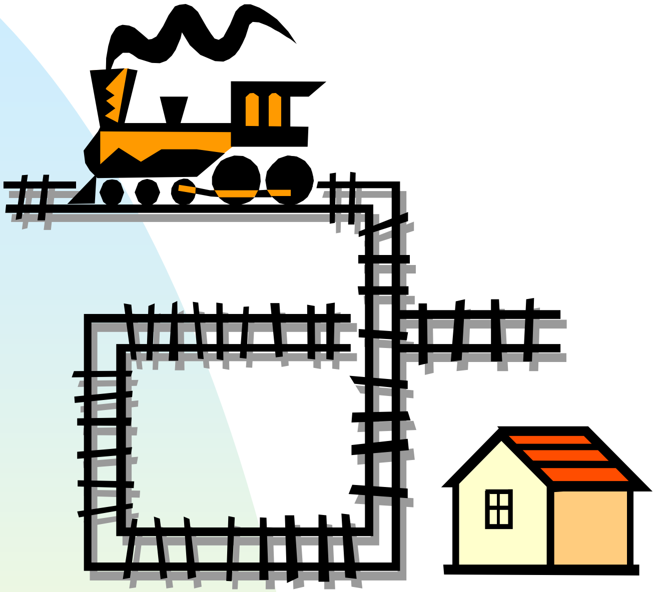
- Combination of *Petri Nets* and *Programming Language.*
  - ***Control structures***, ***synchronisation***, ***communication***, and ***resource sharing*** are described by *Petri Nets.*
  - ***Data*** and ***data manipulations*** are described by *functional programming language.*

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

- Coloured Petri Nets is developed at ***University of Aarhus***, *Denmark* over the last 25 years.



Why do we make models?



- We make *models* to:
  - *Learn new things* about a system.
  - To check that the system design has certain *expected properties.*
- *CPN models* are *dynamic:*
  - They can be *executed* on a *computer.*
  - This allows us to play and *investigate* different *scenarios.*

Overview of talk

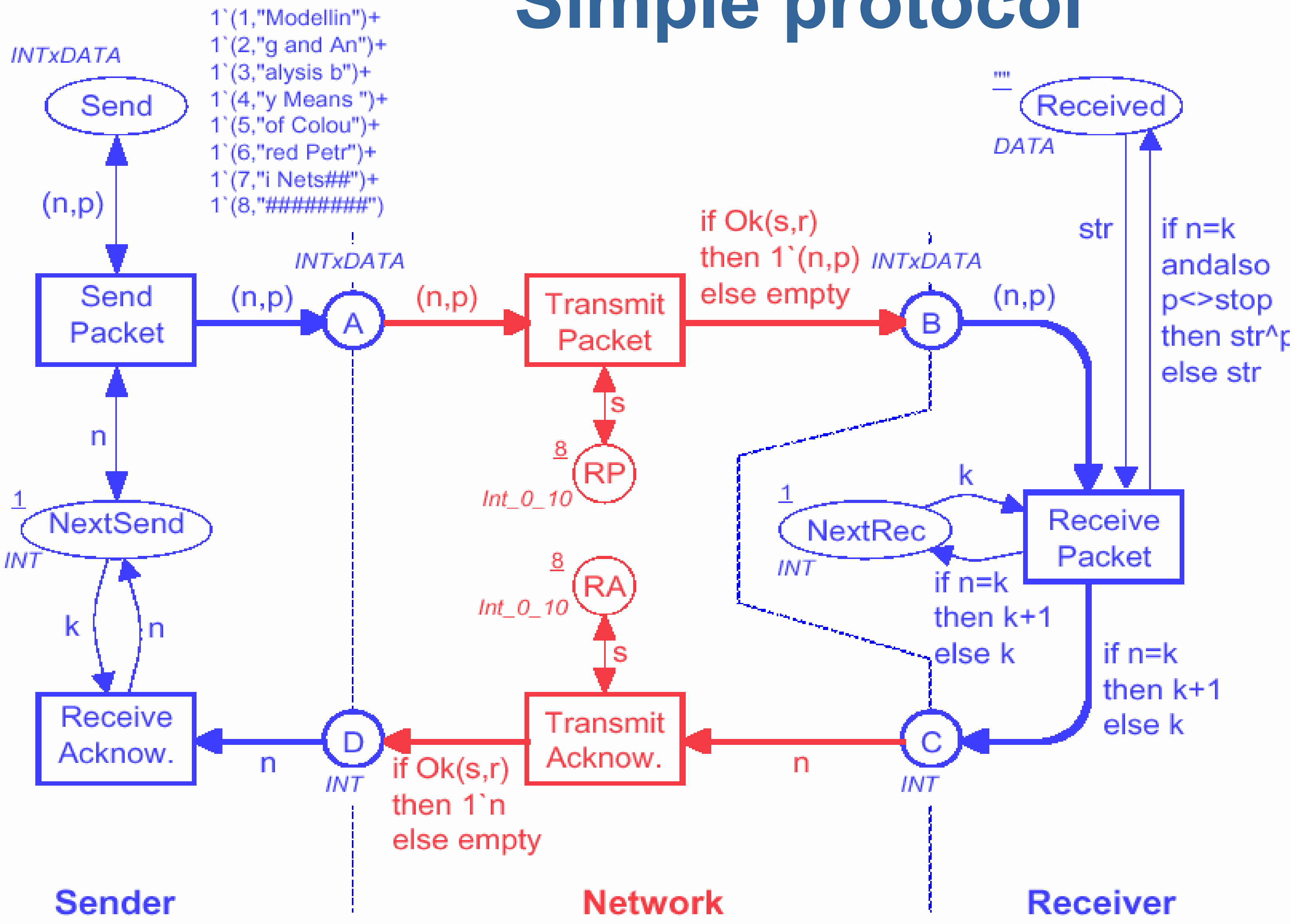
Modelling

- Basic language
  - syntax
  - semantics
- Extensions
  - modules
  - time
- Tool support
  - editing
  - simulation

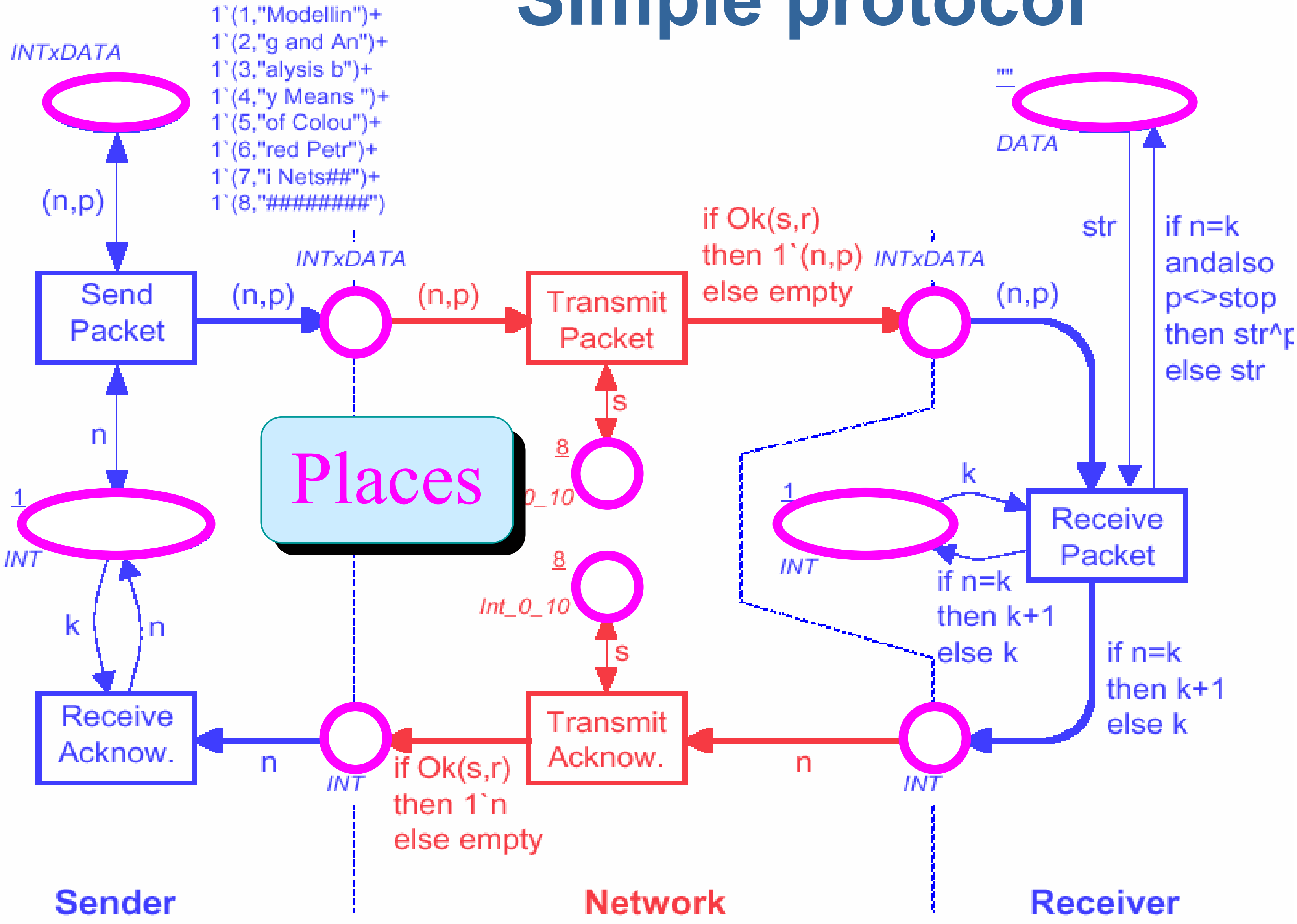
Analysis

- State spaces
  - full
  - symmetries
  - equivalence classes
  - sweep-line
- Place invariants
  - check of invariants
  - use of invariants

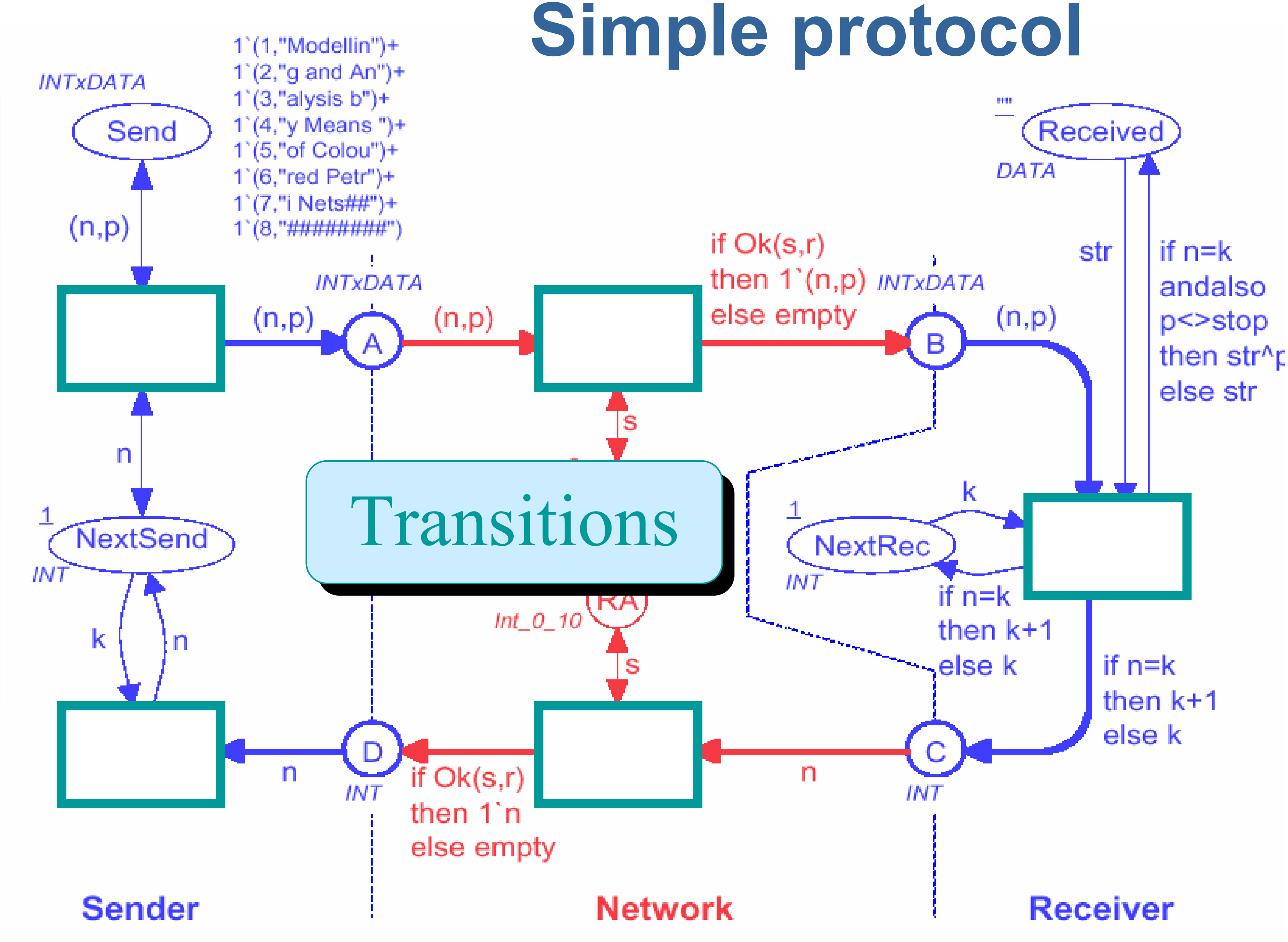
Simple protocol



Simple protocol

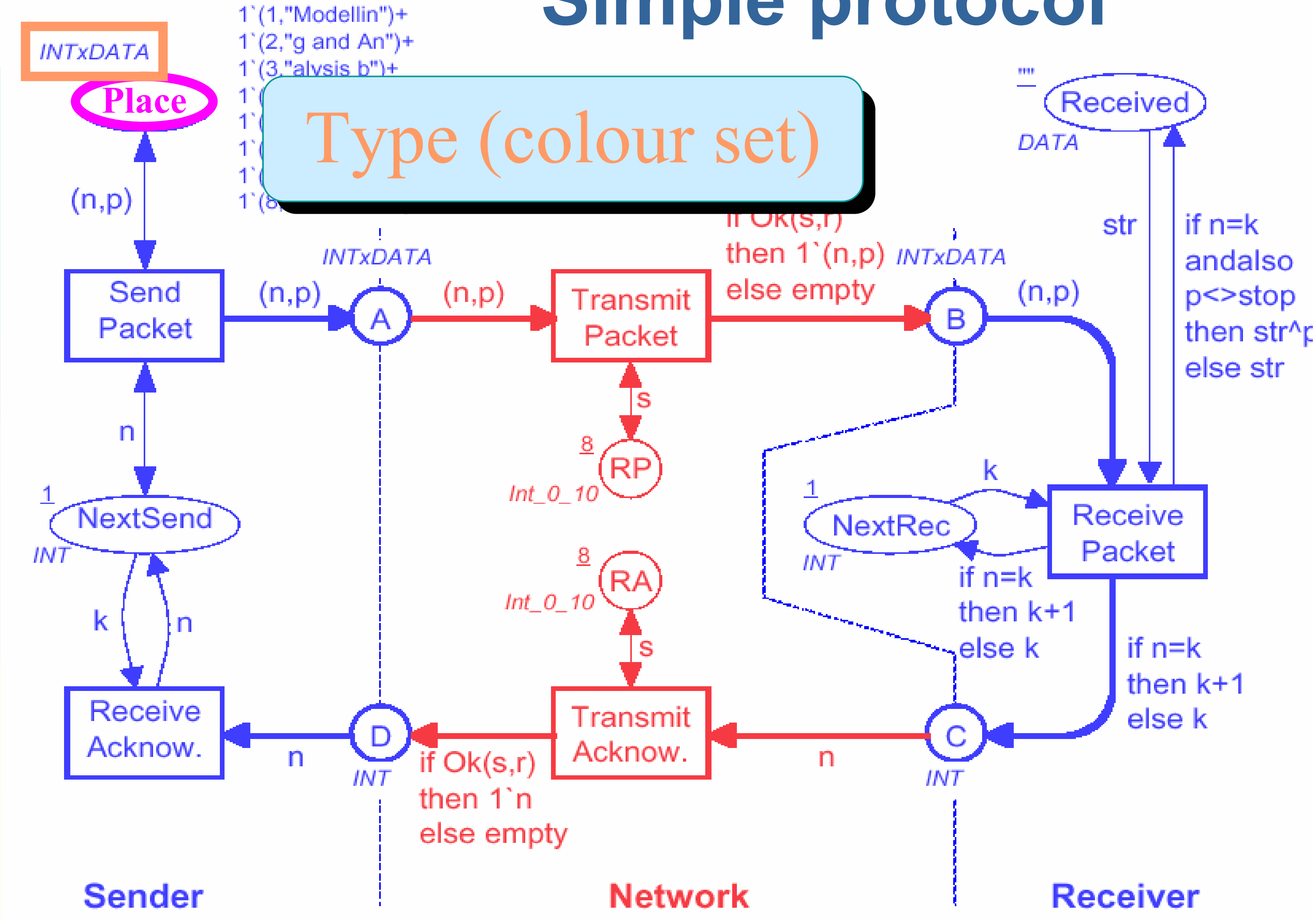


Simple protocol



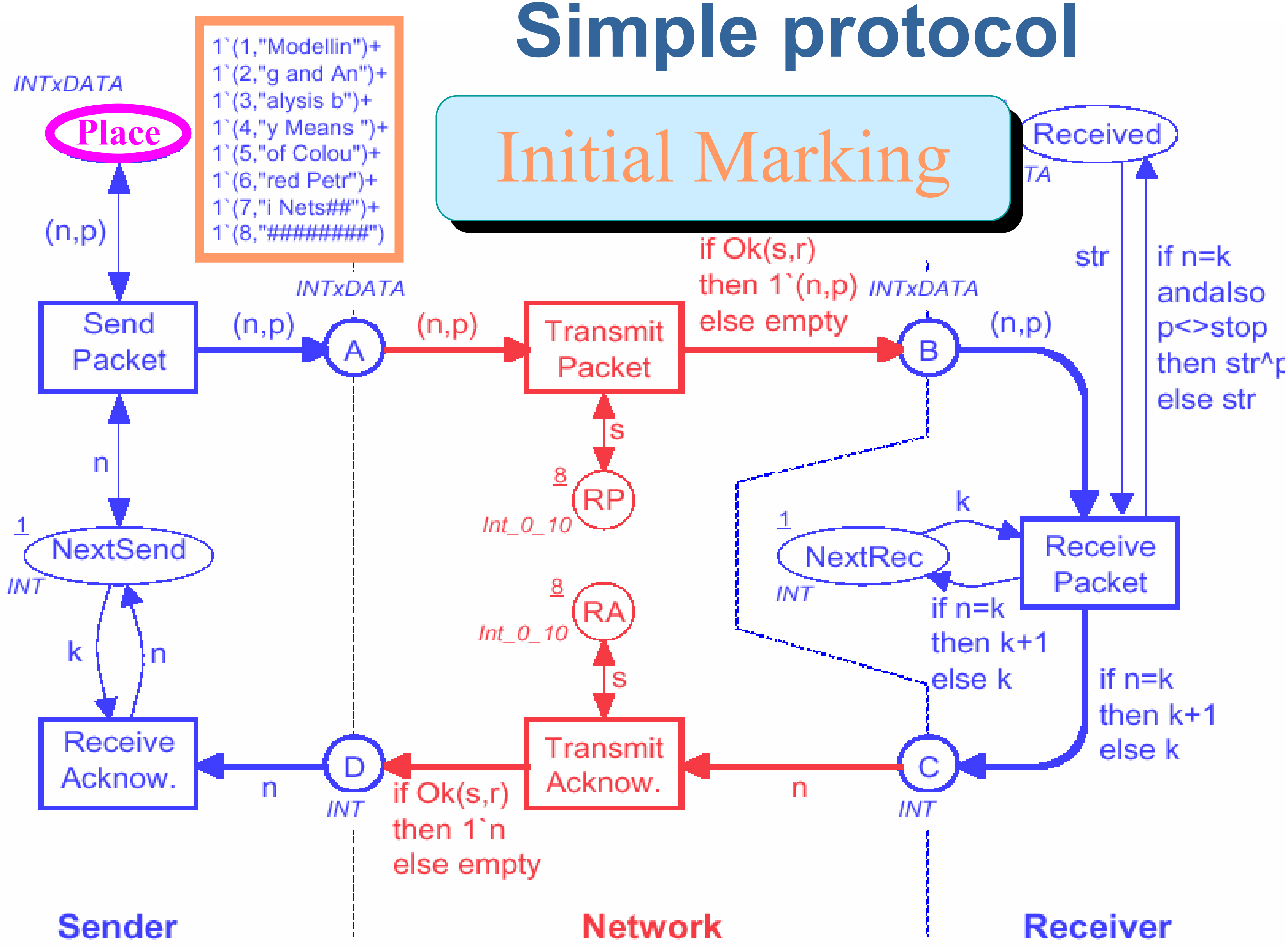
Transitions

Simple protocol



Type (colour set)

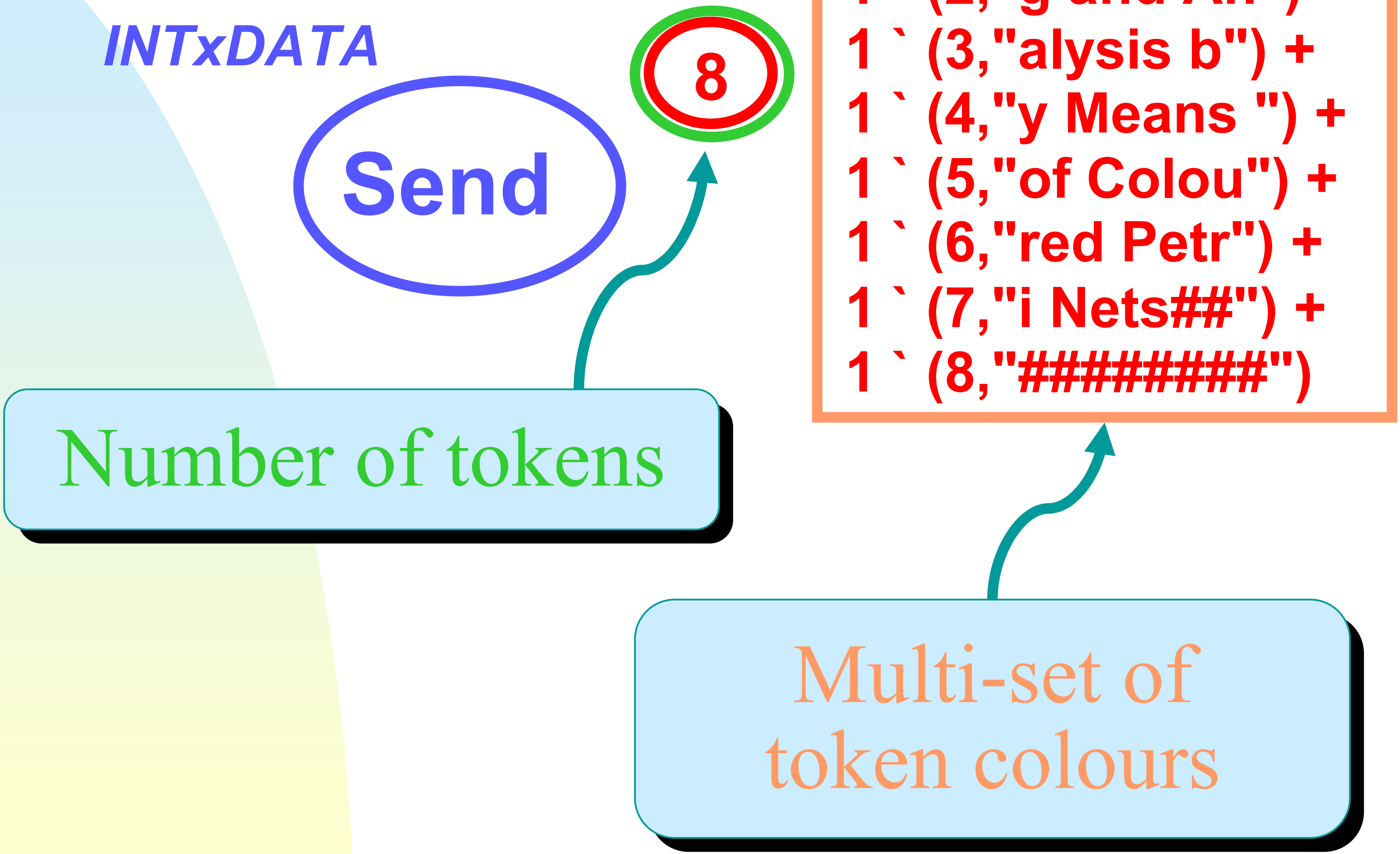
Simple protocol



Initial Marking

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,"########")

Marking of Send



INTxDATA

Send

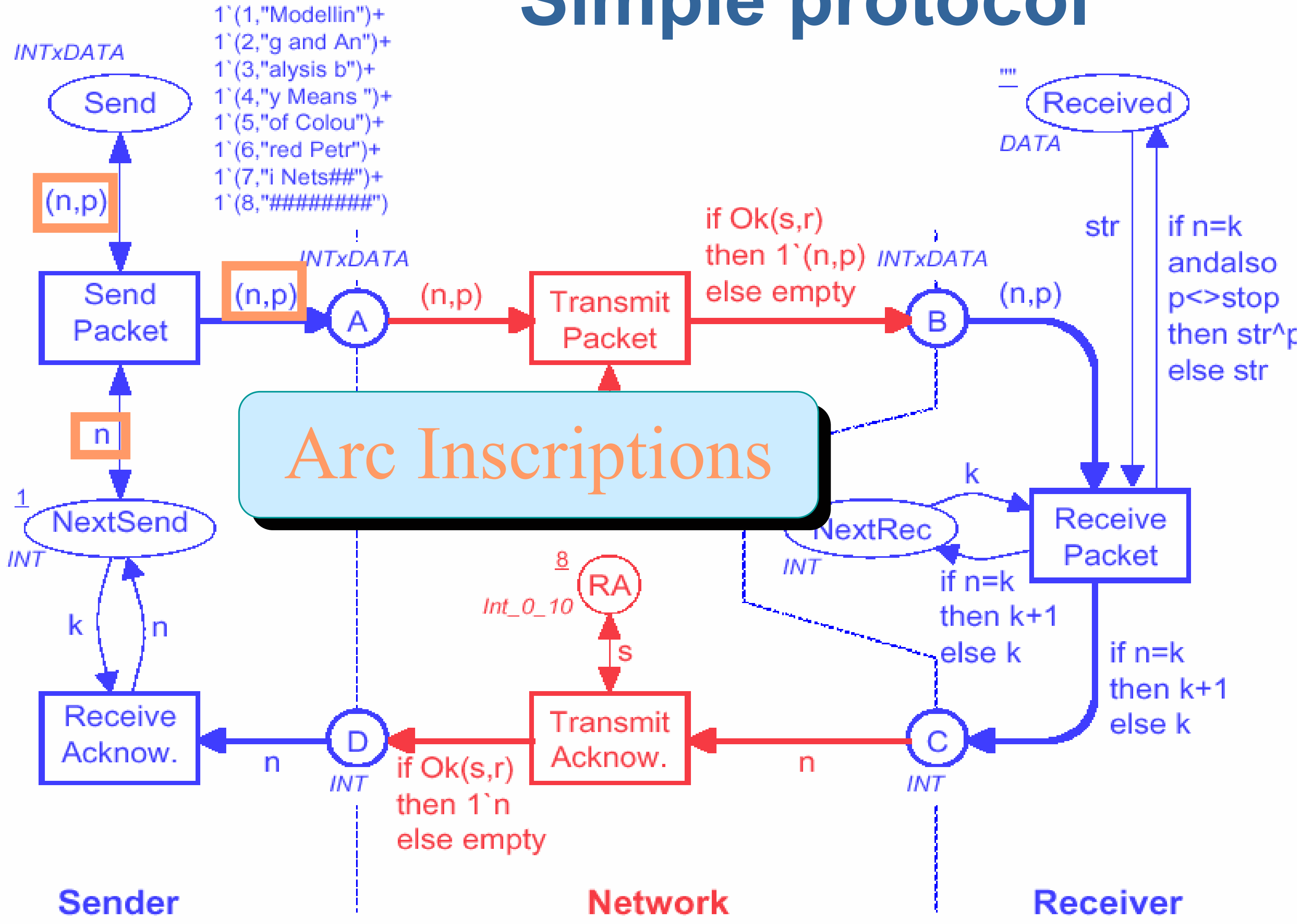
8

```
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,"########")
```

Number of tokens

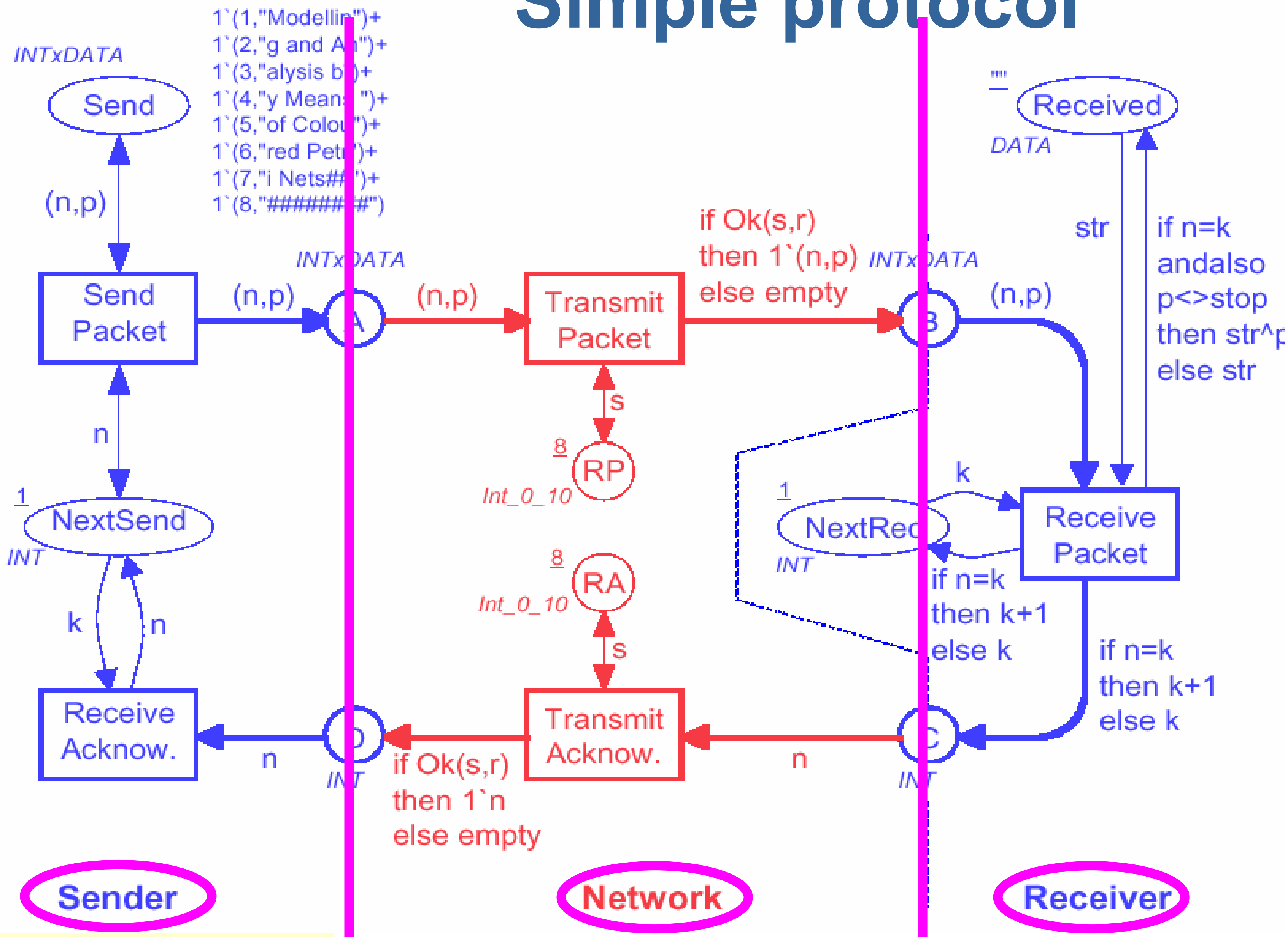
Multi-set of token colours

Simple protocol

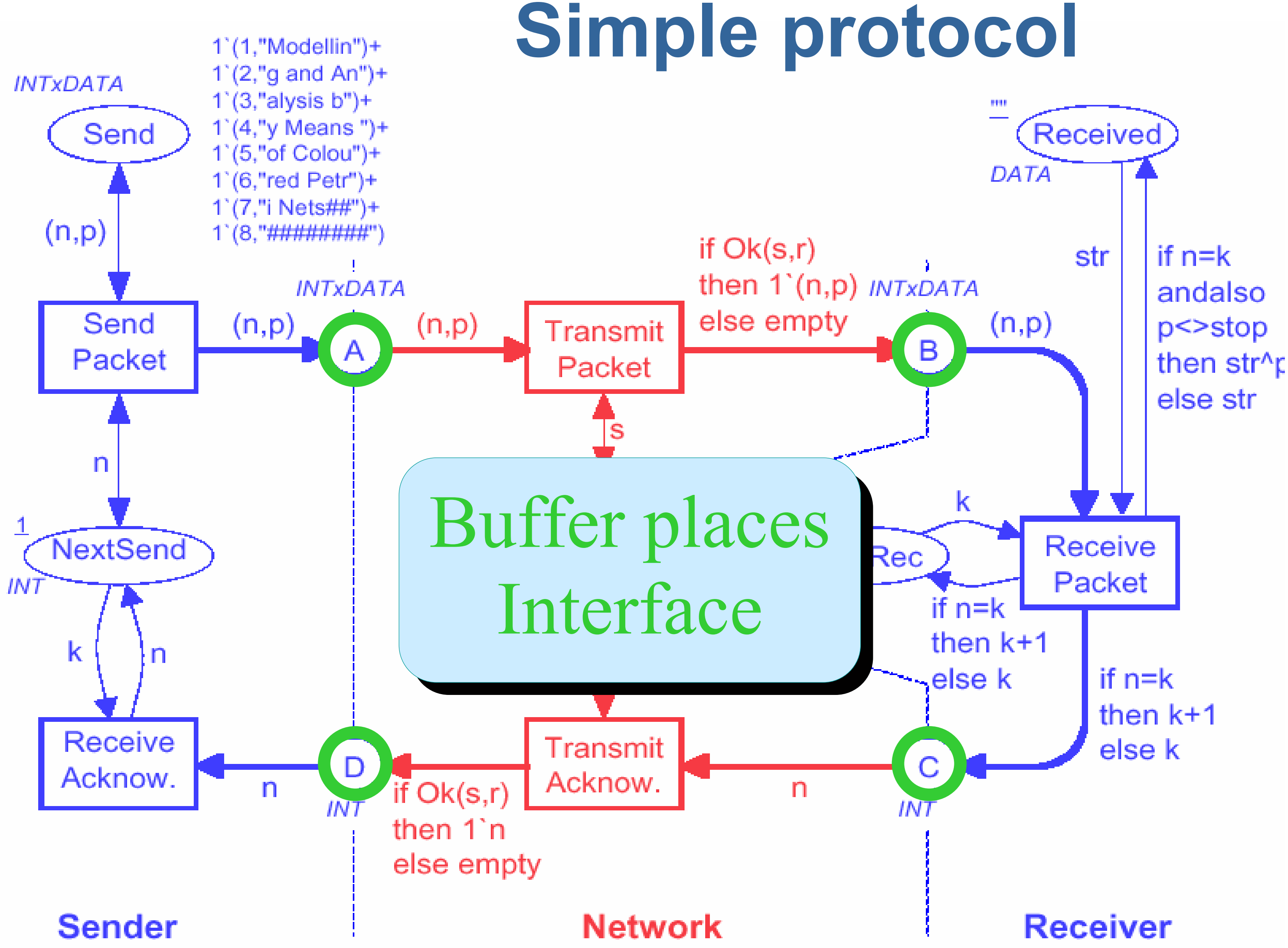


Arc Inscriptions

Simple protocol

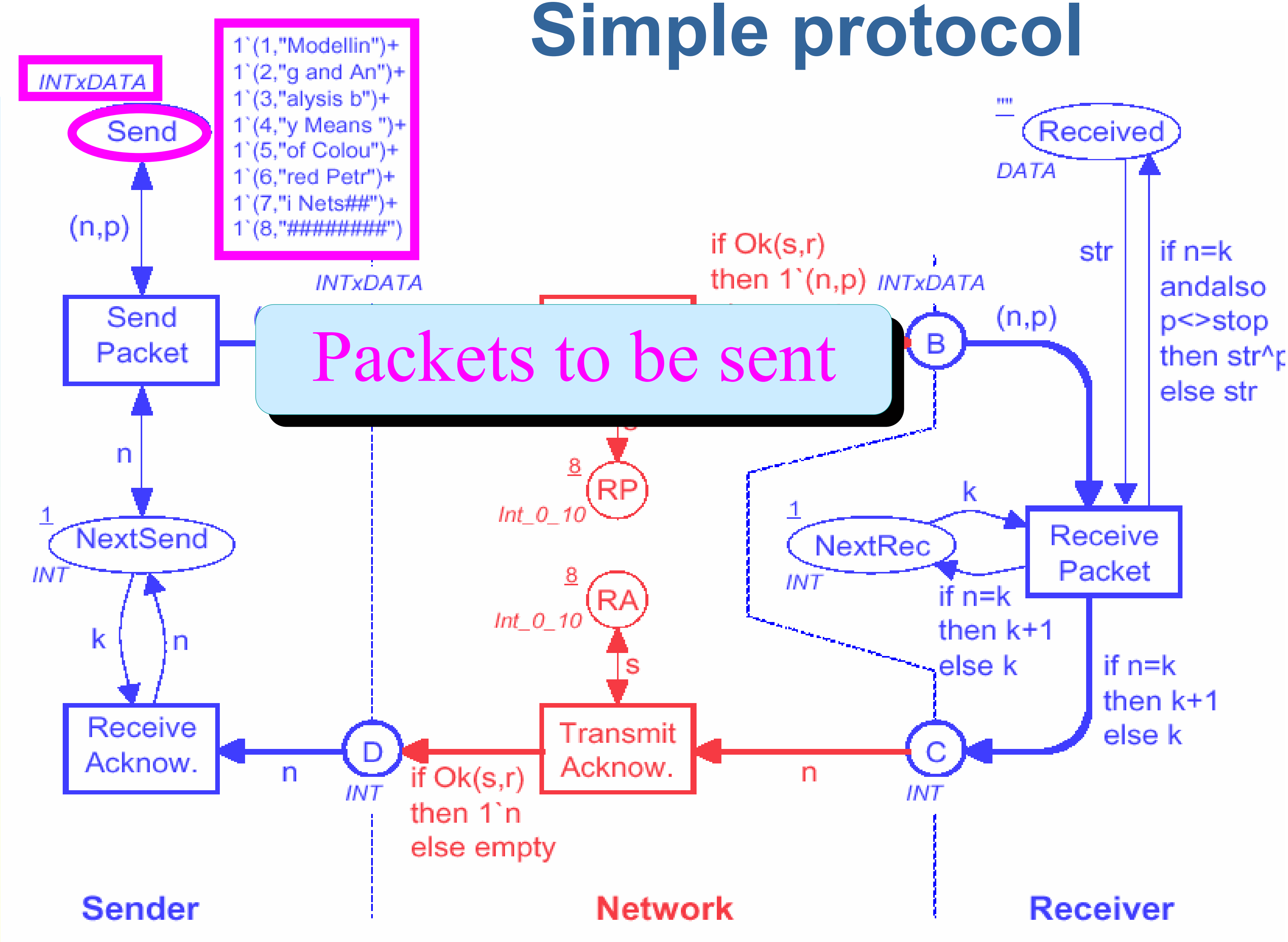


Simple protocol



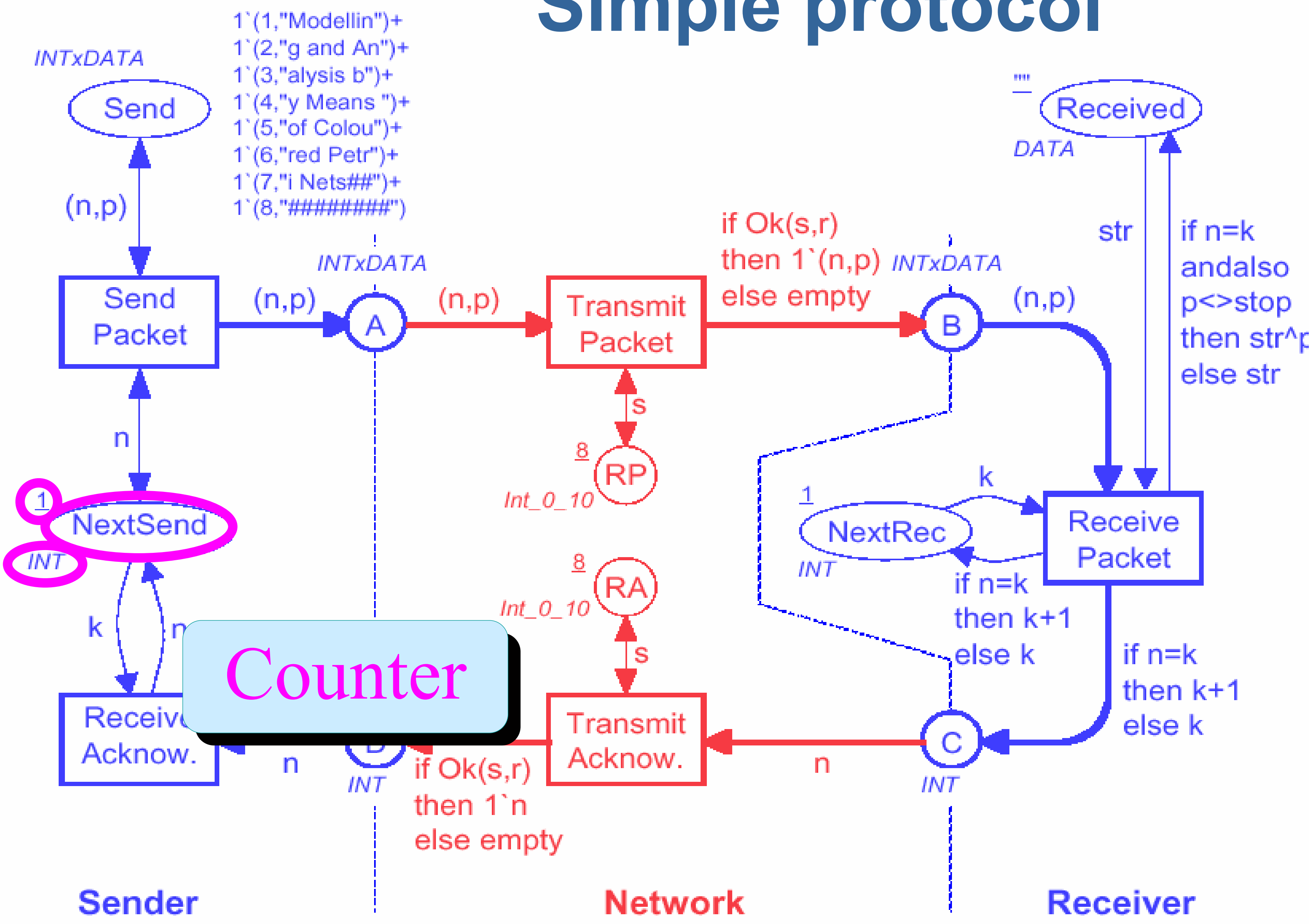
Buffer places Interface

Simple protocol



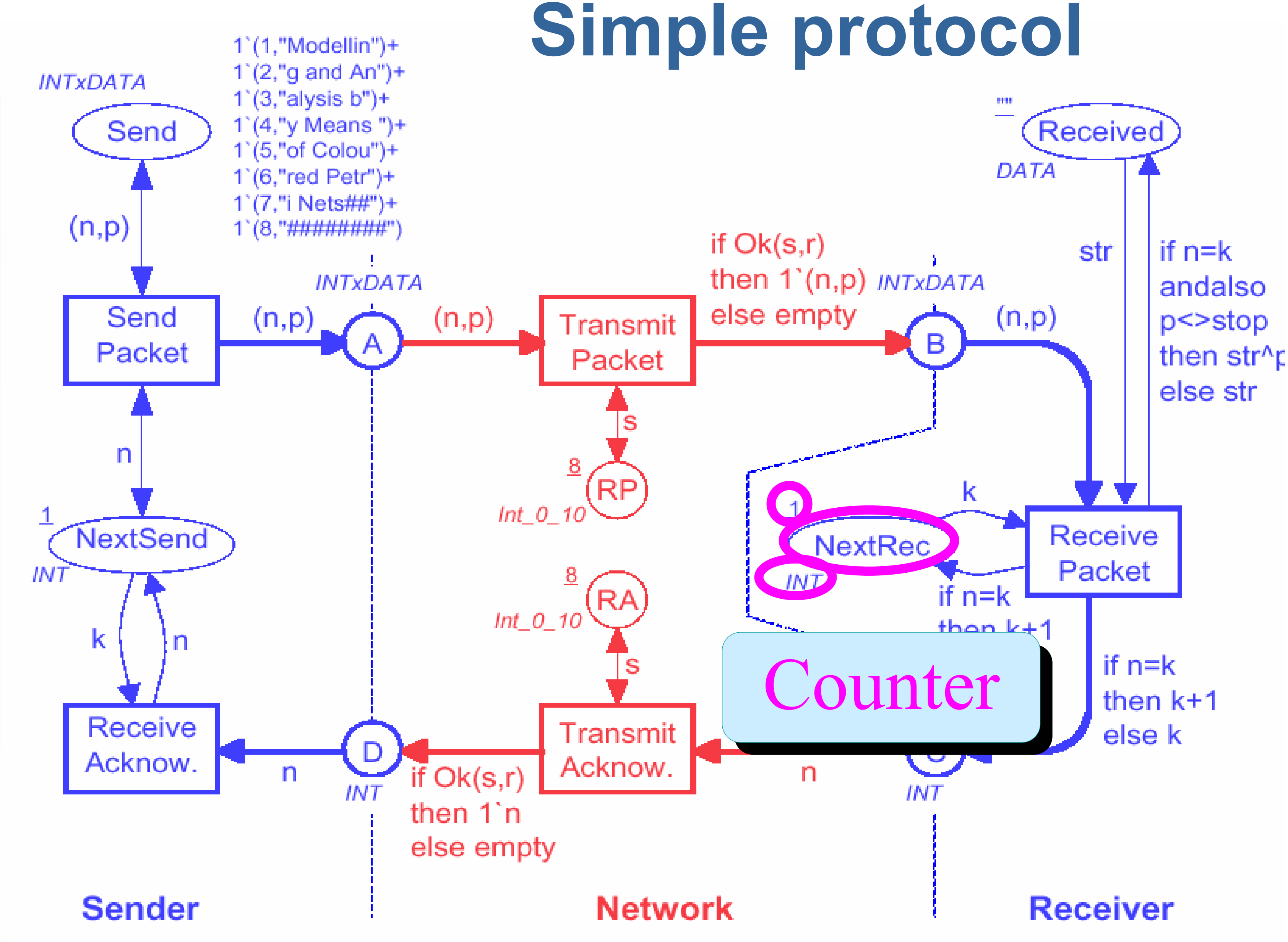
Packets to be sent

Simple protocol

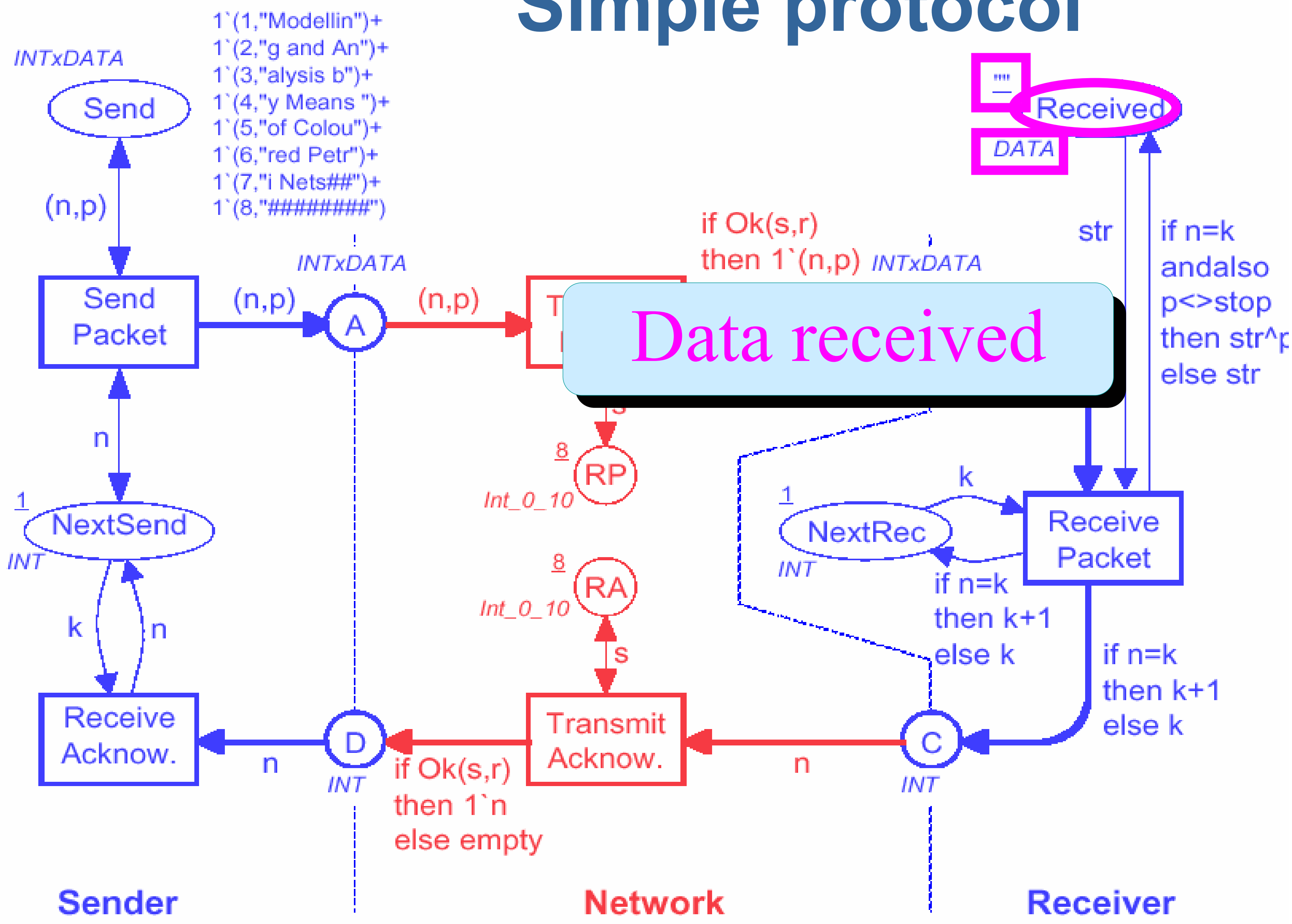


Counter

Simple protocol

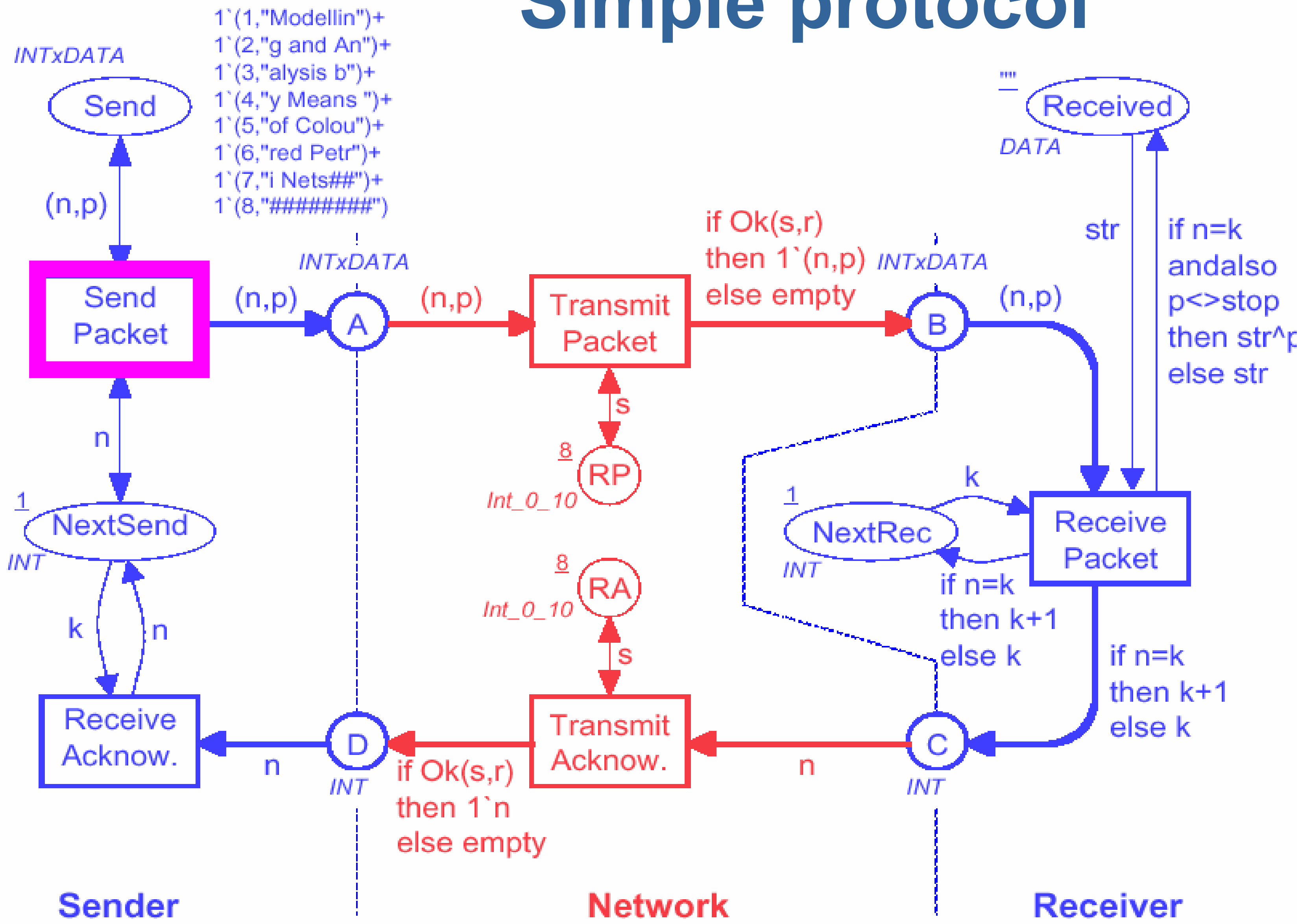


Simple protocol



Data received

Simple protocol



Send packet

- The binding

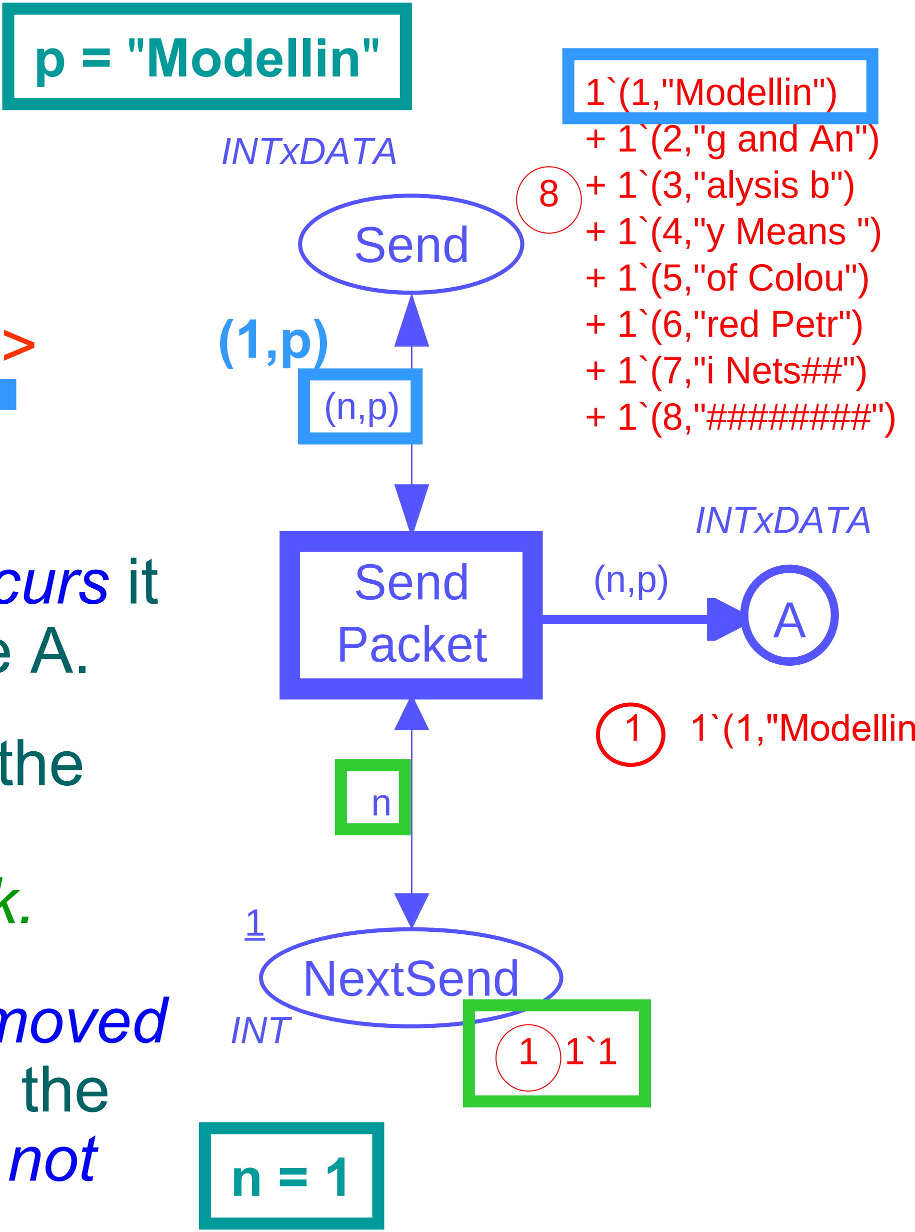
  <n=1,p="Modellin">

  is *enabled.*

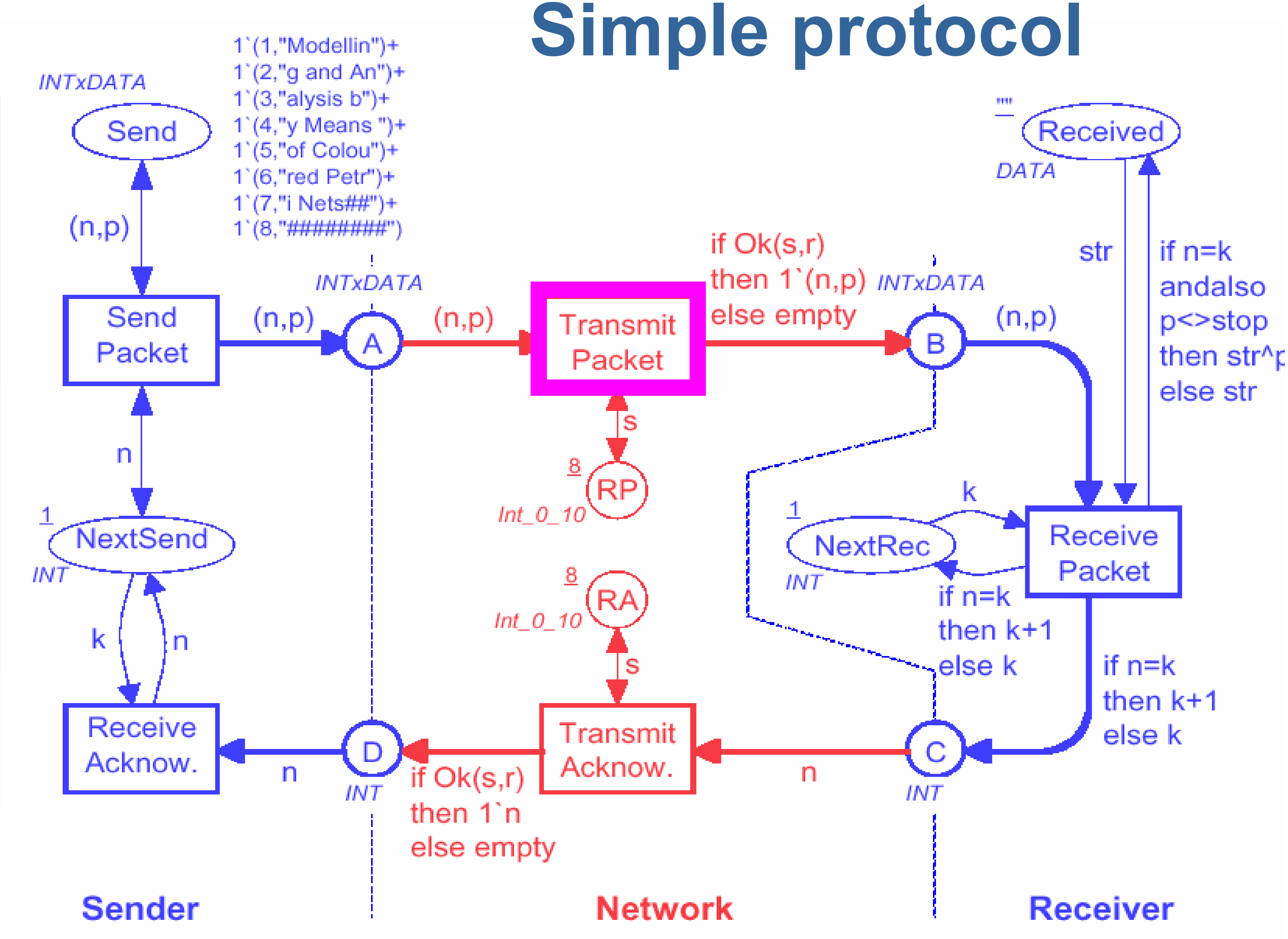
- When the binding *occurs* it *adds a token* to place A.

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

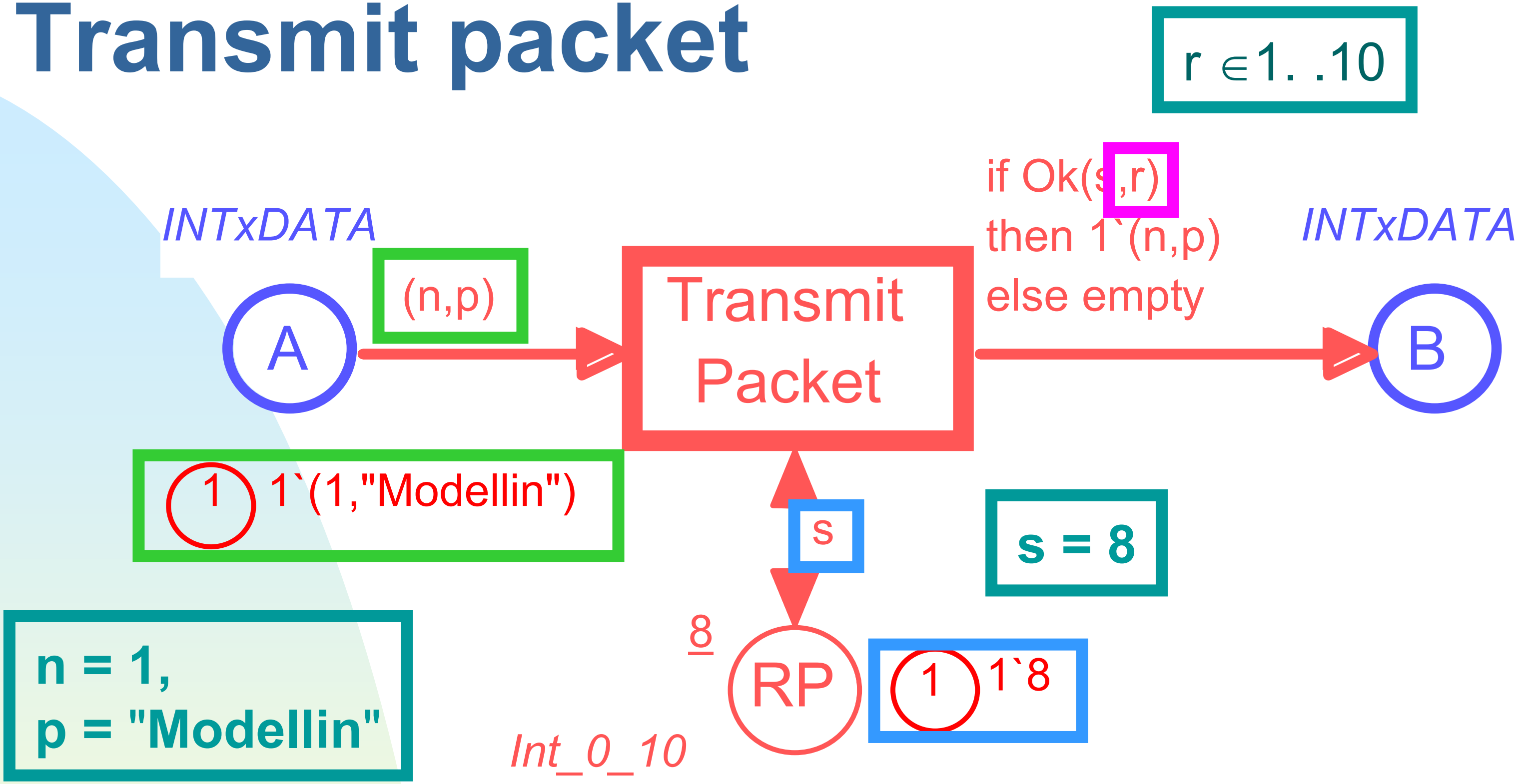
- The packet is *not removed* from place *Send* and the *NextSend* counter is *not changed.*



Simple protocol



Transmit packet



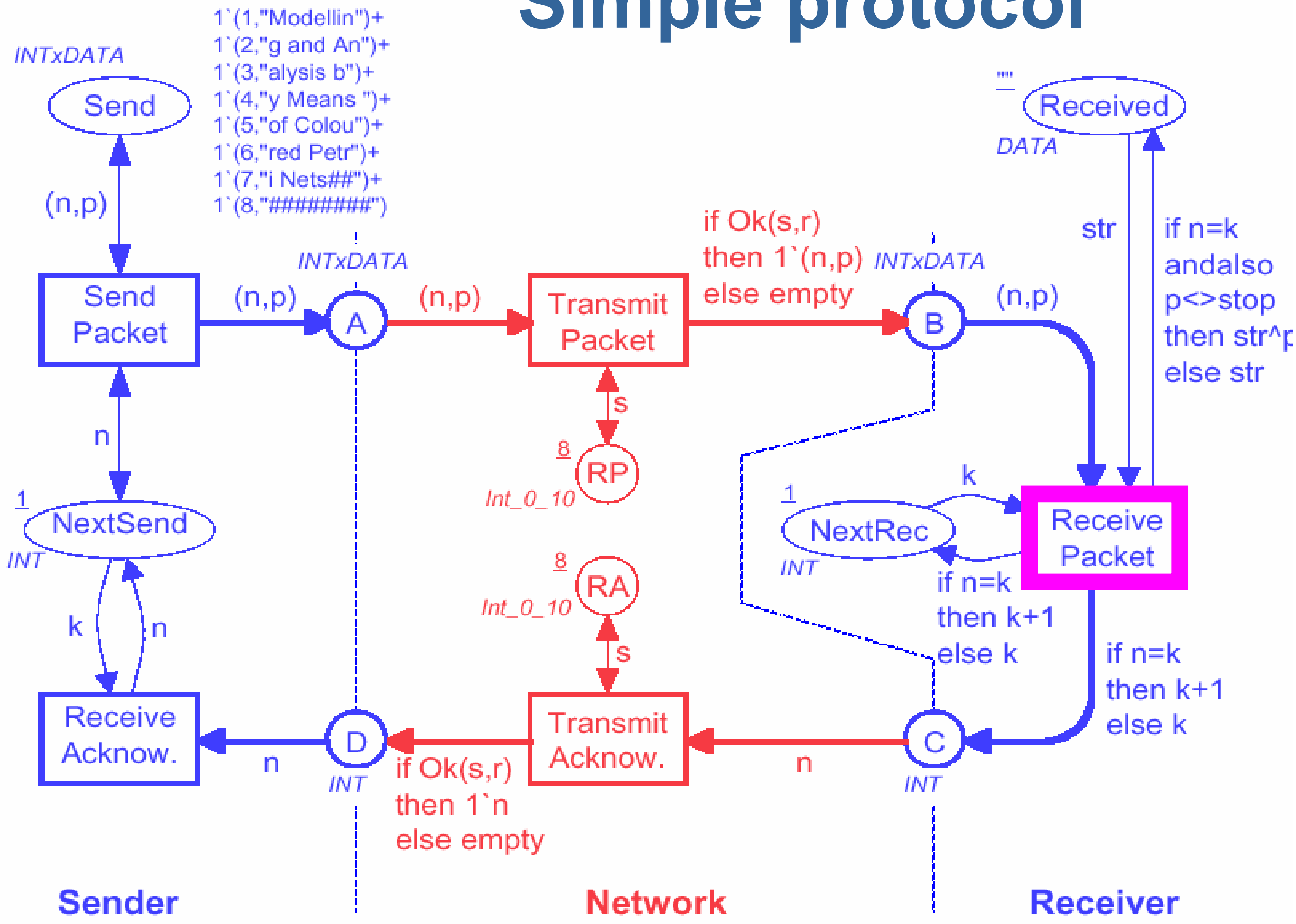
- All *enabled bindings* are on the form:
  - <n=1,p= "Modellin",s=8,r=...>
  - where r ∈1. .10

Loss of packets

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

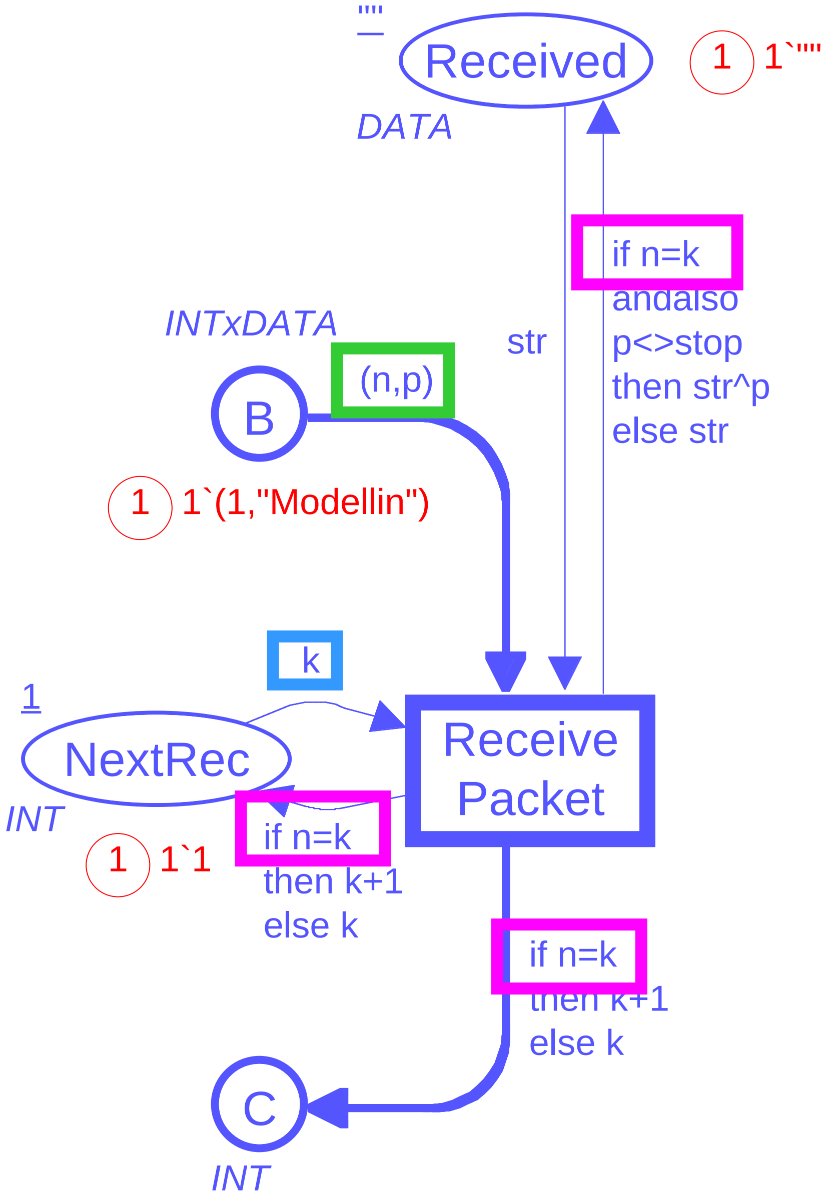
- The *function* *Ok(s,r)* checks whether *$r \leq s$.*
  - *For $r \in 1..8$, Ok(s,r)=true.*
    The token is moved from A to B. This means that the packet is *successfully transmitted* over the network.
  - *For $r \in 9..10$, Ok(s,r)=false.*
    No token is added to B. This means that the packet is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

Simple protocol



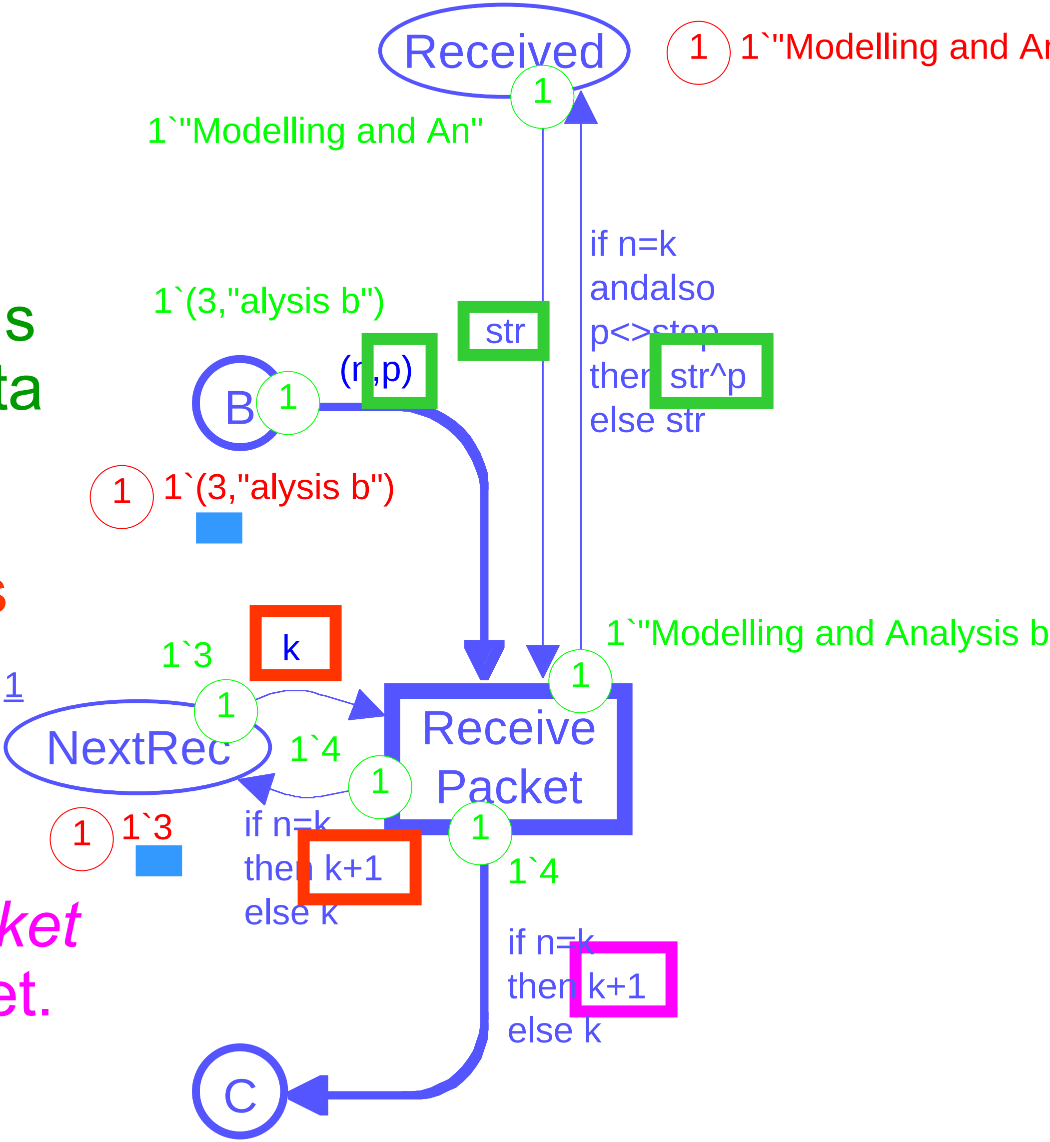
Receive packet

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



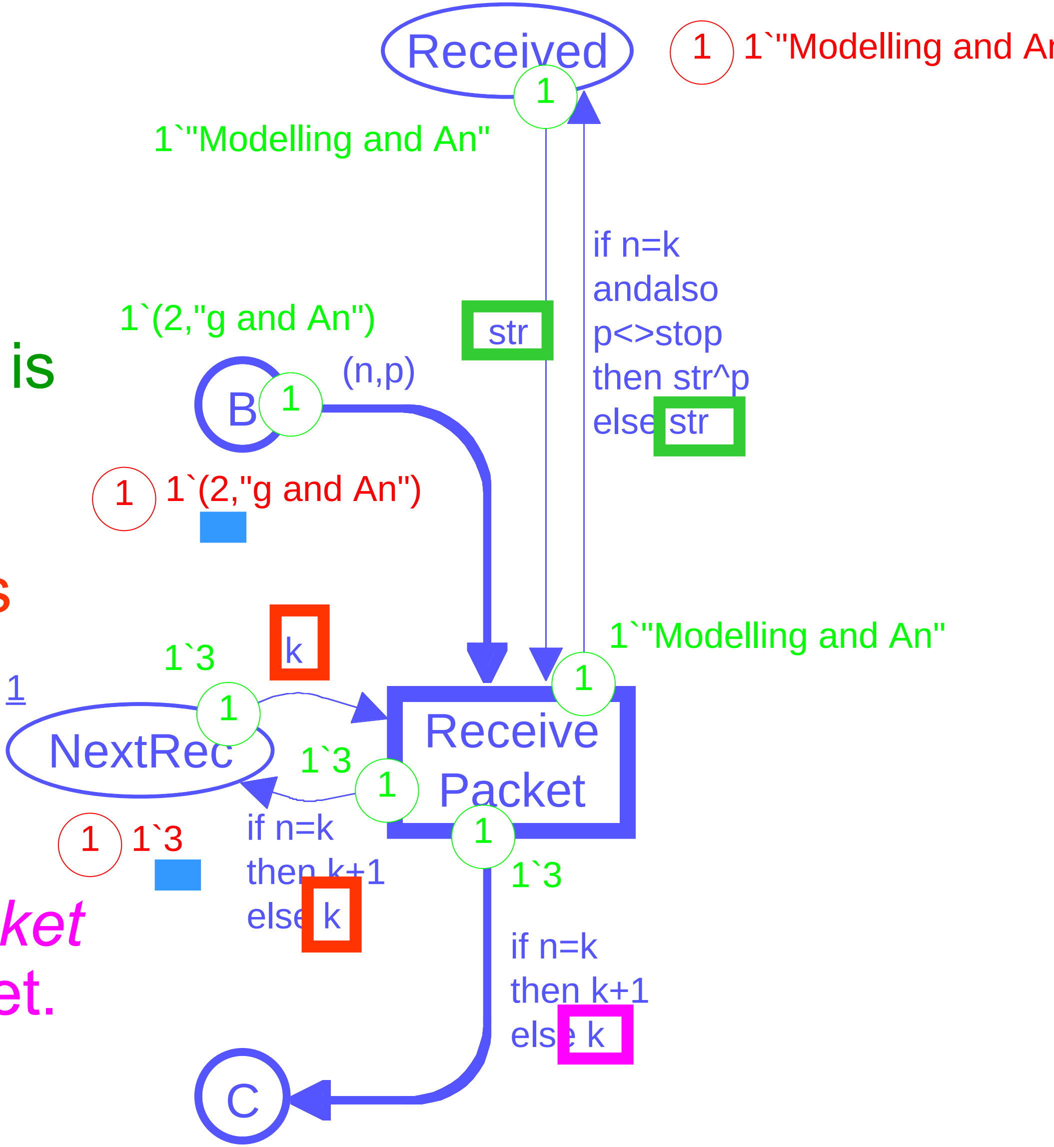
Correct packet number

- The data in the packet is *concatenated* to the data already received.
- The *NextRec* counter is *increased by one.*
- An *acknowledgement* is sent. It contains the number of the *next packet* the receiver wants to get.

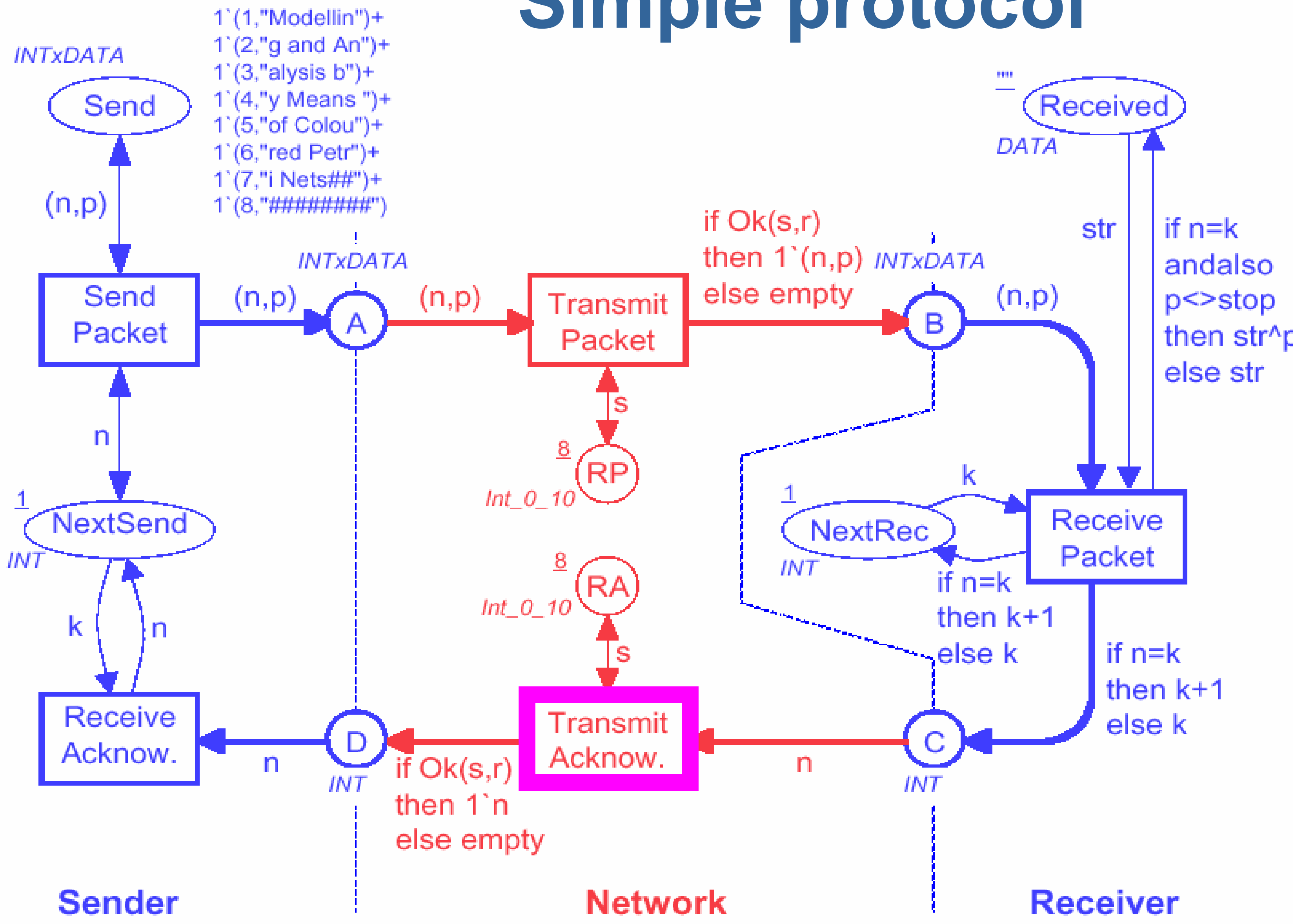


Wrong packet number

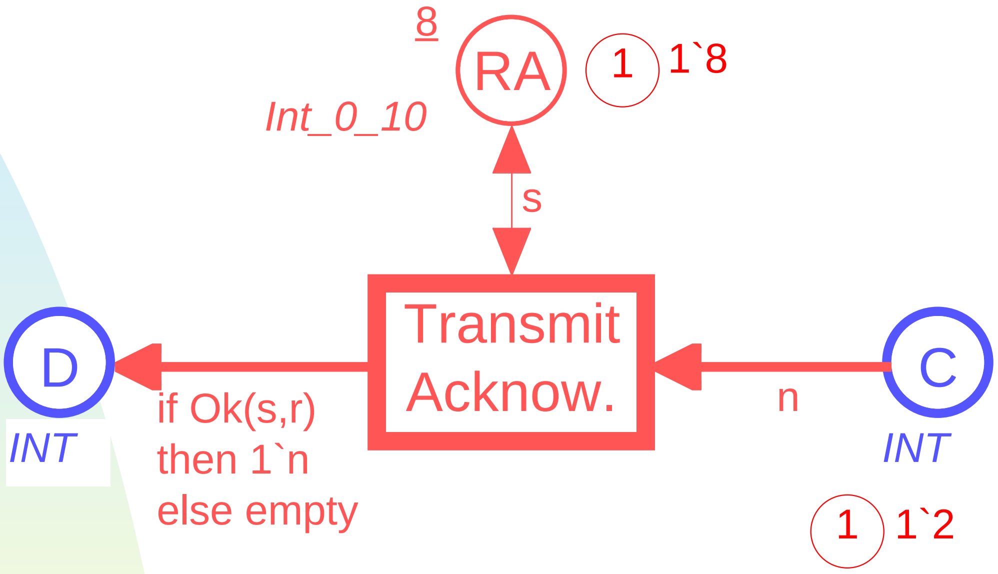
- The data in the packet is *ignored.*
- The *NextRec* counter is *unchanged.*
- An *acknowledgement* is sent. It contains the number of the *next packet* the receiver wants to get.



Simple protocol

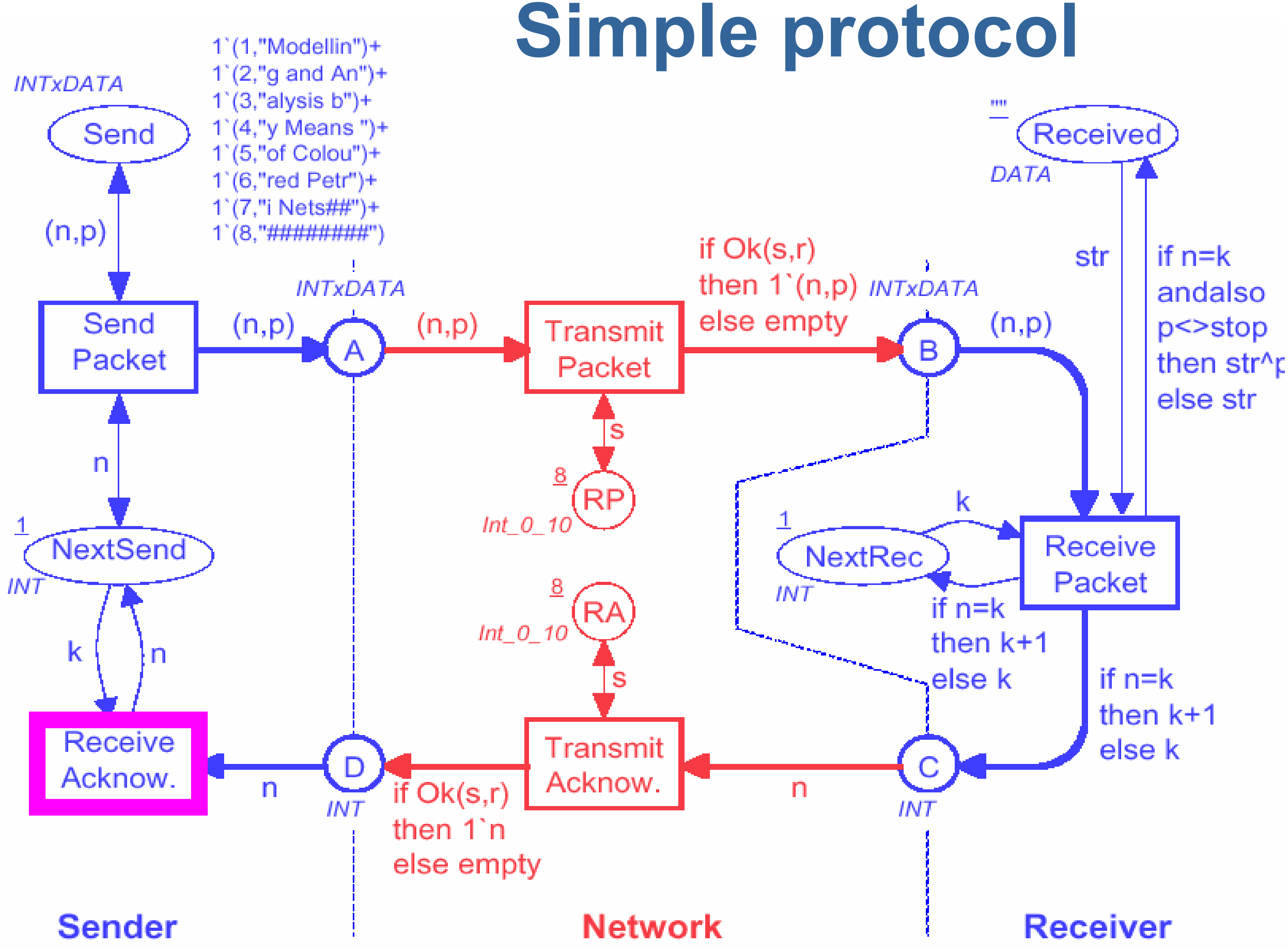


Transmit acknowledgement

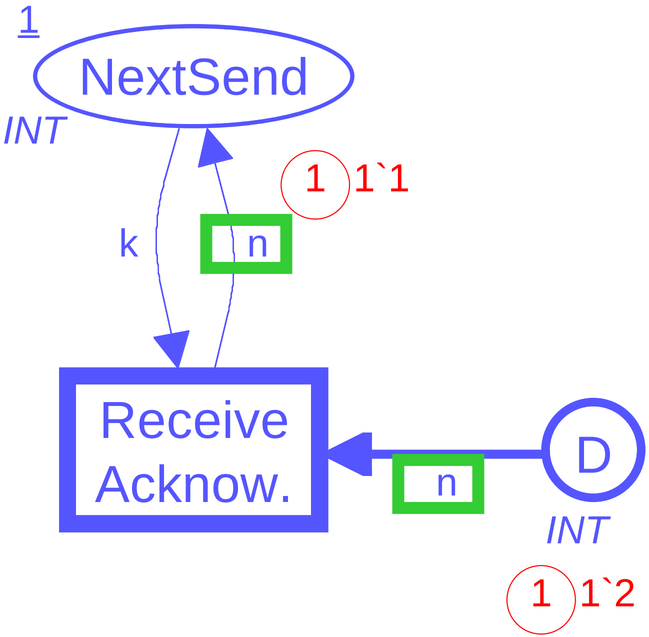


- This transition works in a similar way as *Transmit Packet.*
- The marking of *RA* determines the *success rate.*

Simple protocol



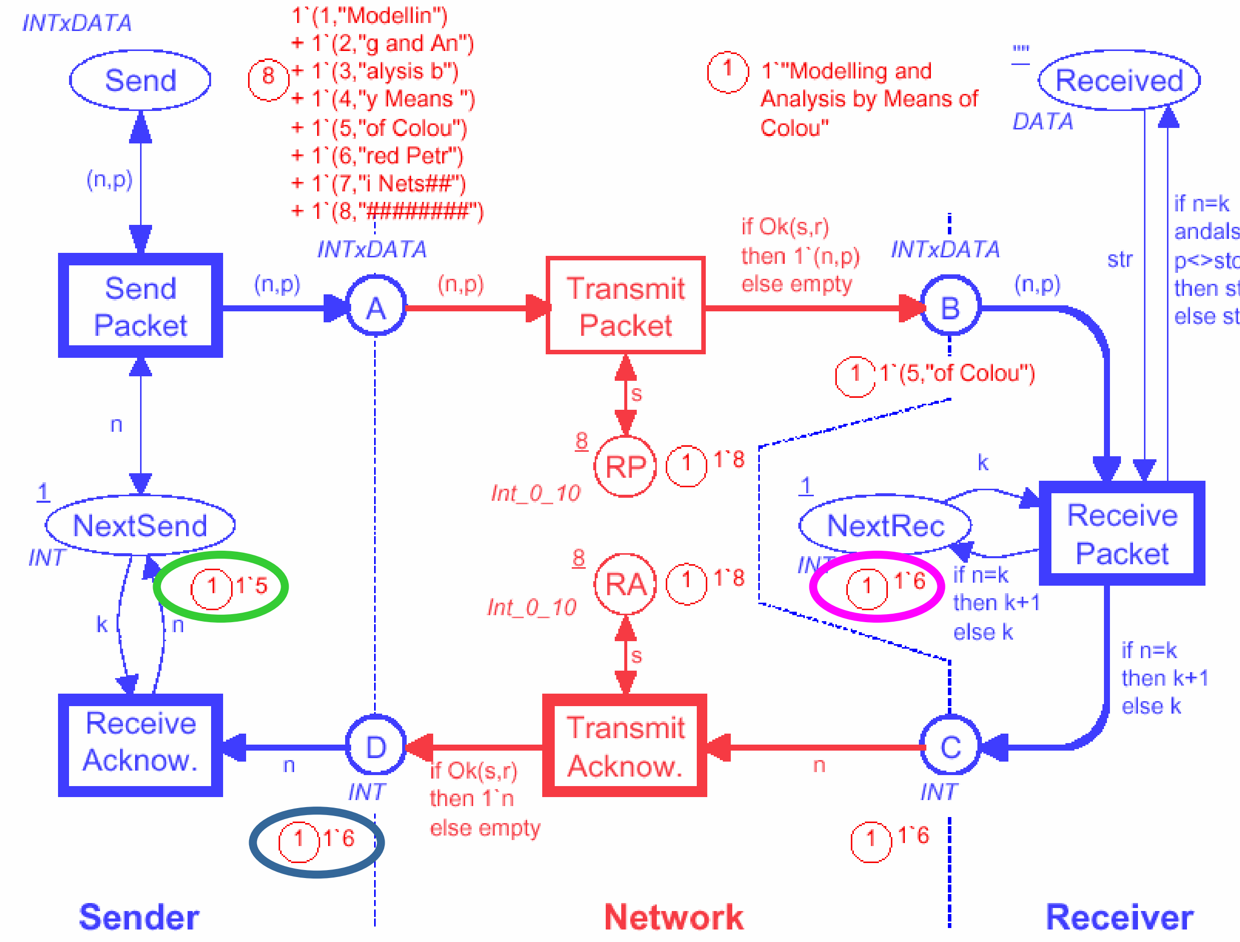
Receive acknowledgement



- 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 state

- *Receiver* expects packet no. 6.
- *Sender* is still sending packet no. 5.
- *Acknowledgement* requesting packet no. 6 is arriving.
- Then *NextSend* is updated and *Sender* will start sending packet no. 6.



CP-nets has a formal definition

- The existence of a *formal definition* is important:
  - Basis for *simulation*, i.e., execution of the CP-net.
  - Basis for the *formal verification* methods (e.g., state spaces and place invariants).
  - Without the formal definition, it would have been impossible to obtain a *sound* net class.
- It is *not necessary* for a *user* to know the formal definition of CP-nets:
  - Correct *syntax* is checked by the CPN editor.
  - Correct *semantics* is guaranteed by the CPN simulator and the CPN verification tools.

High-level Petri nets

- The relationship between *CP-nets* and *ordinary Petri nets* (PT-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.*

Overview of talk

Modelling

- Basic language
  - syntax
  - semantics
- **Extensions**
  - modules
  - time
- Tool support
  - editing
  - simulation

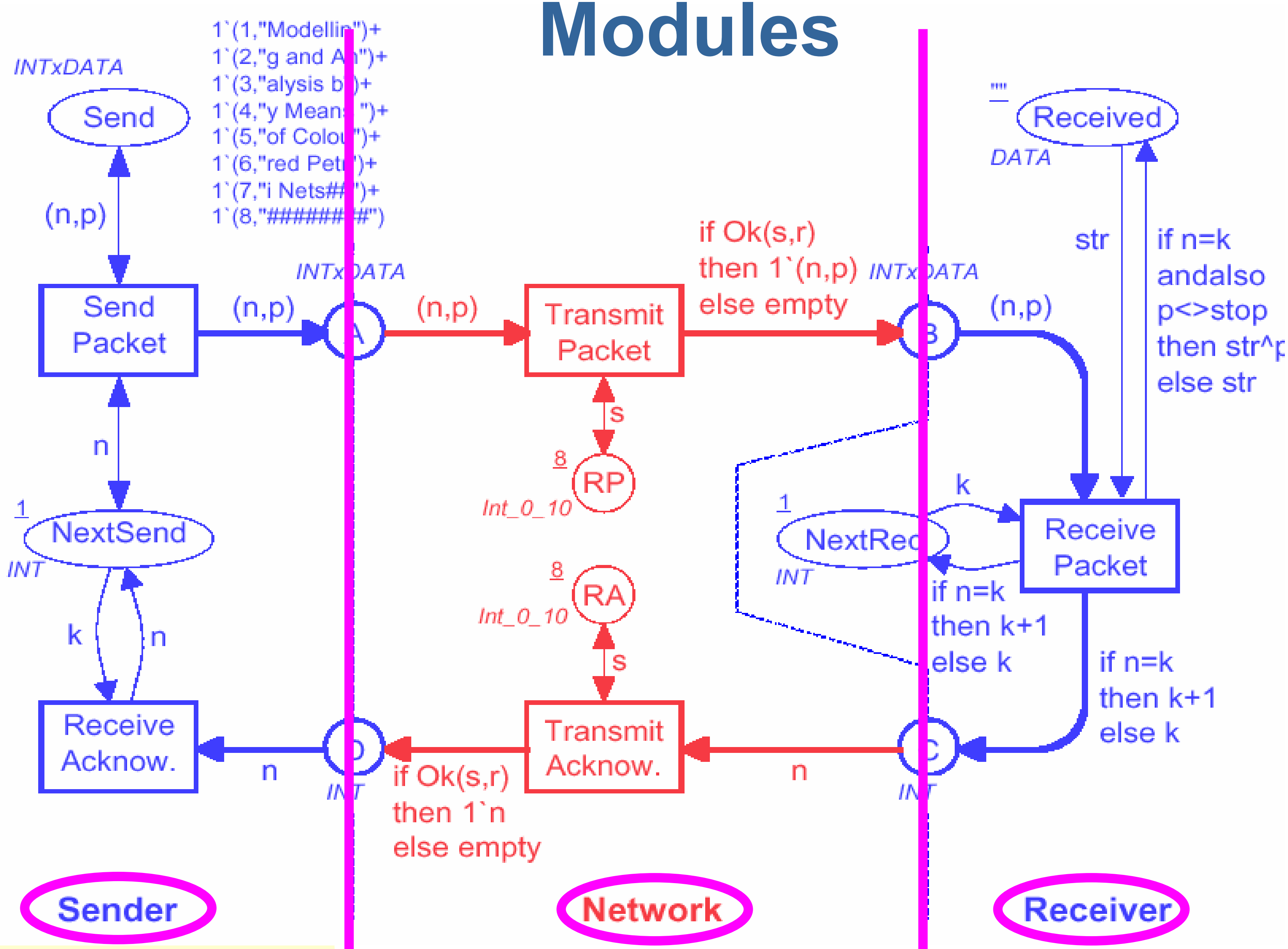
Analysis

- State spaces
  - full
  - symmetries
  - equivalence classes
  - sweep-line
- Place invariants
  - check of invariants
  - use of invariants

CP-nets are used for large systems

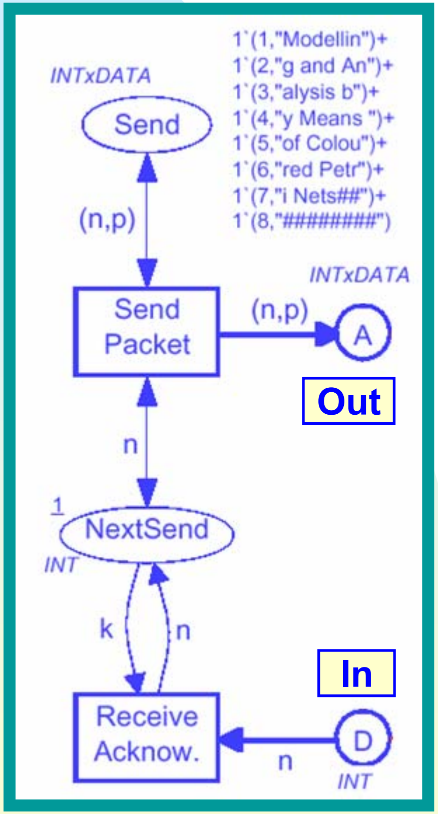
- A CPN model consists of a number of *modules*.
  - Also called *subnets* or *pages.*
  - Well-defined *interfaces* and clear *semantics.*
- A typical *industrial application* of CP-nets has:
  - 10-200 modules.
  - 50-1000 places and transitions.
  - 10-200 types.
- Industrial applications of this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

Modules

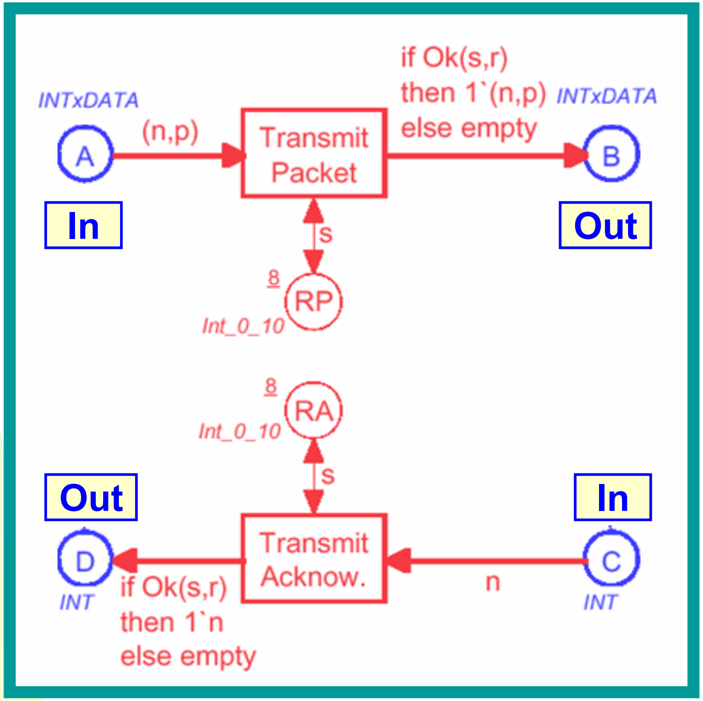

INTxDATA
Send
(n,p)
1`(1,"Modellin")+
1`(2,"g and An")+
1`(3,"alysis b")+
1`(4,"y Means ")+
1`(5,"of Colou")+
1`(6,"red Petr")+
1`(7,"i Nets##")+
1`(8,"########")
Send Packet
(n,p)
INTxDATA
A
(n,p)
Transmit Packet
if Ok(s,r) then 1`(n,p) else empty
INTxDATA
B
(n,p)
""
Received
DATA
str
if n=k andalso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

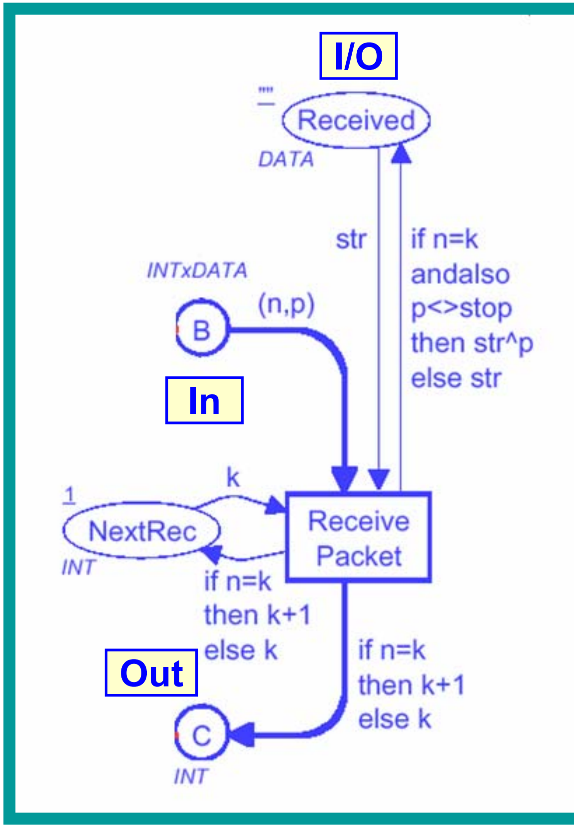
Sender



Network

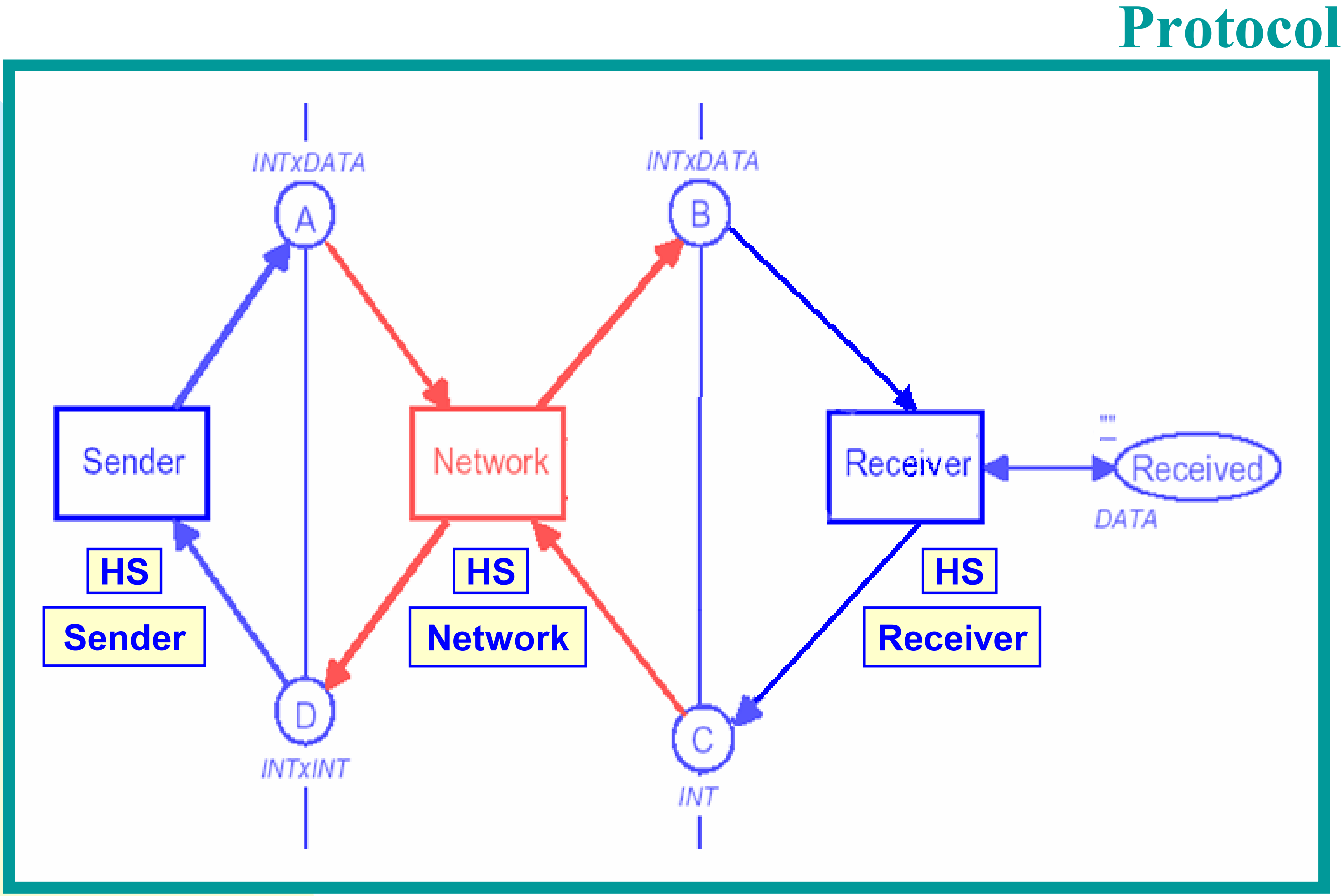


Receiver



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

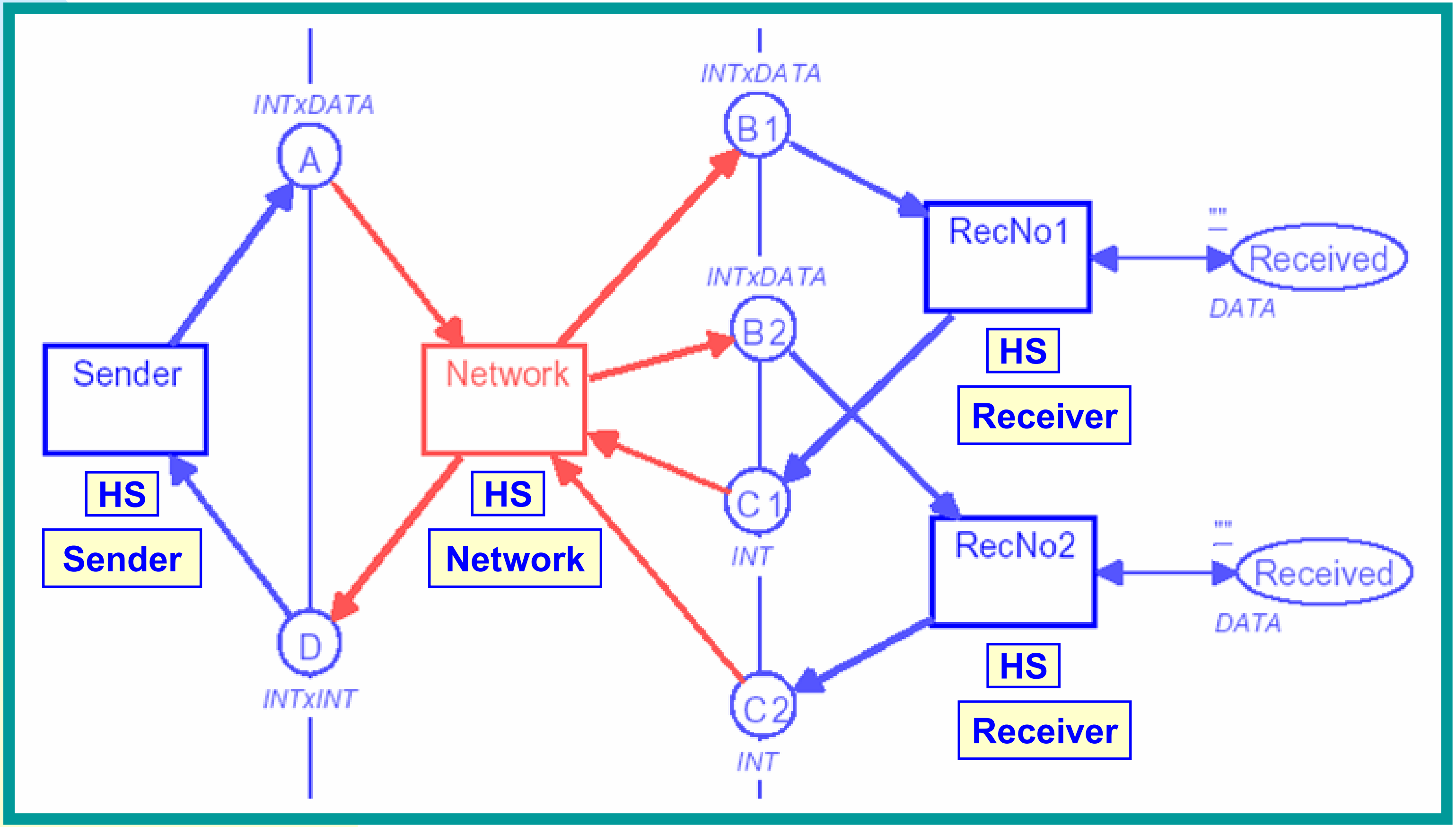
Abstract view



- *Substitution transitions* refer to *modules.*
- *Socket places* are related to *port places.*

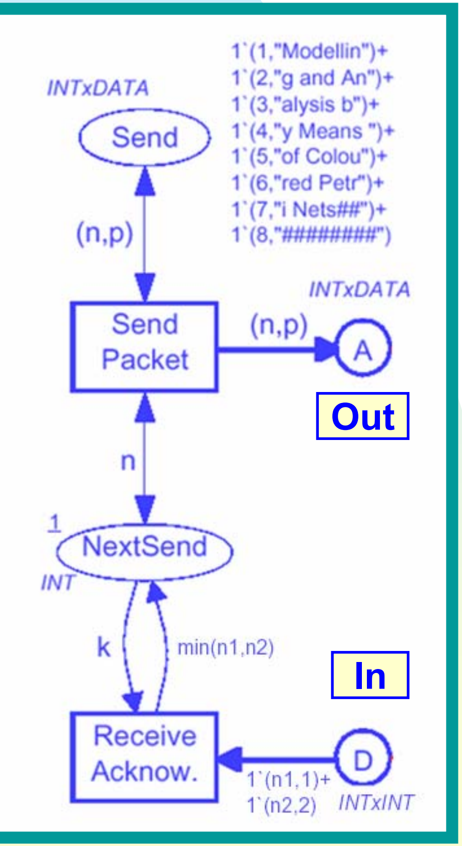
Modules can be reused

Protocol

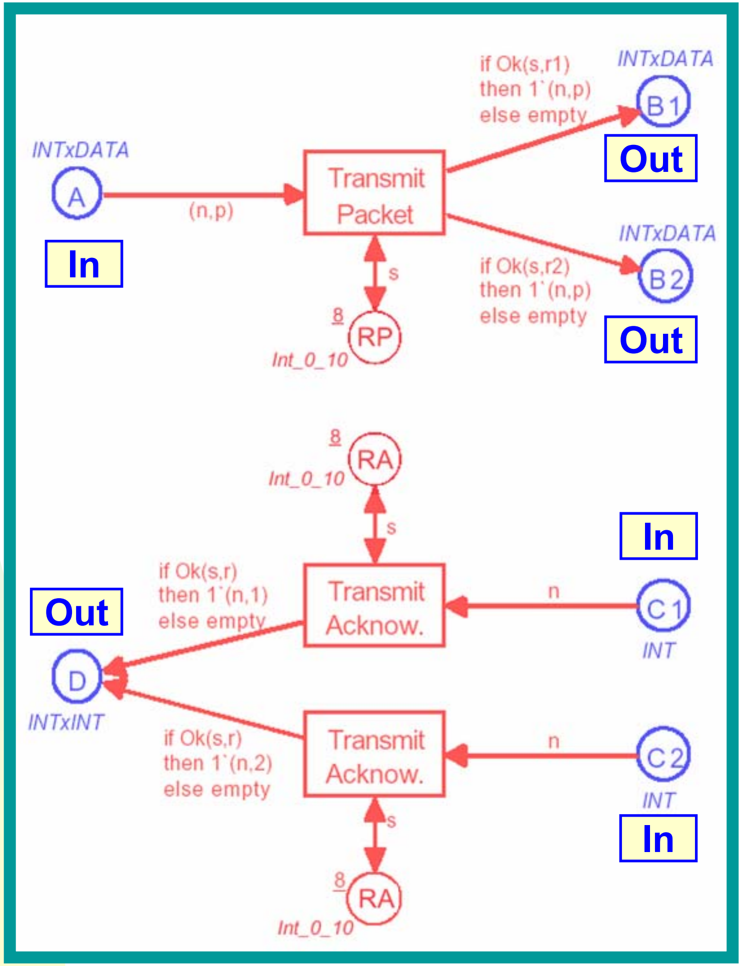


Protocol with multiple receivers

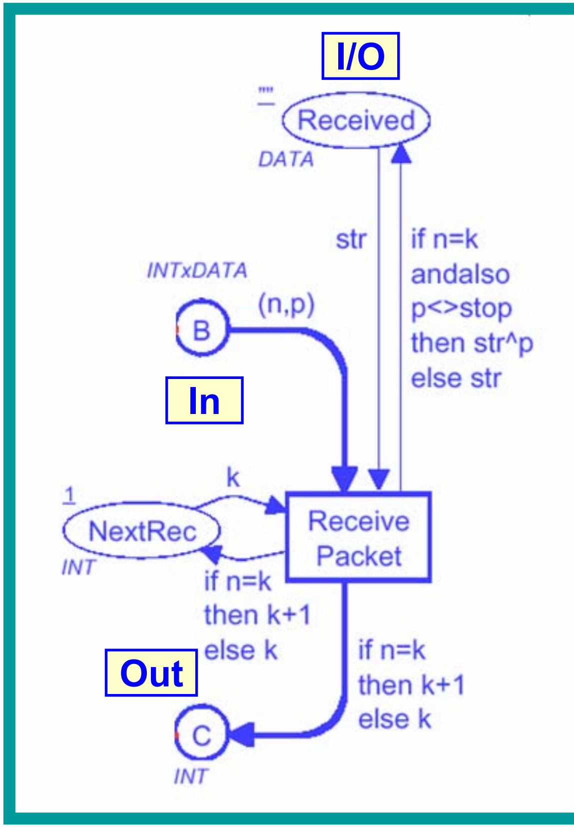
Sender



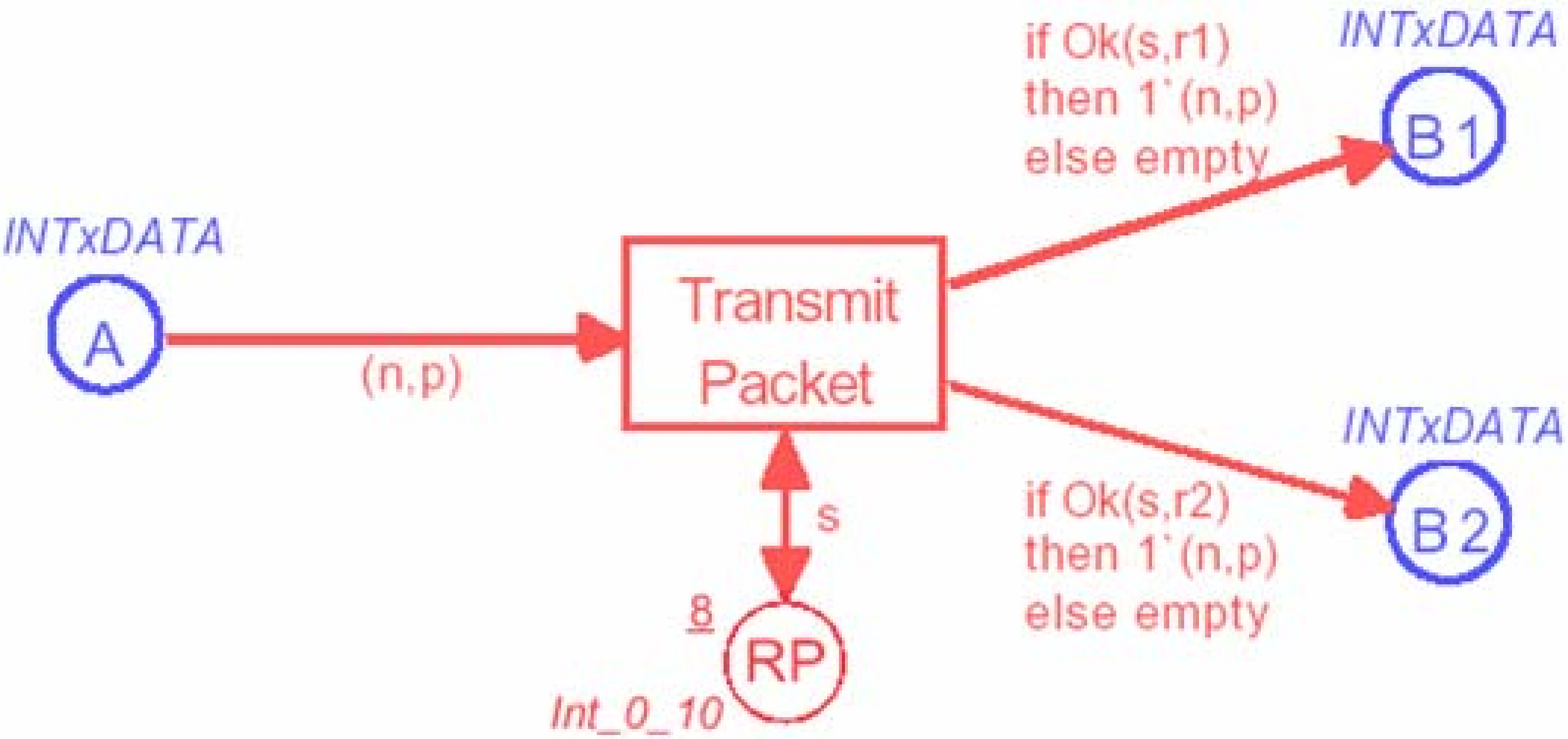
Network



Receiver

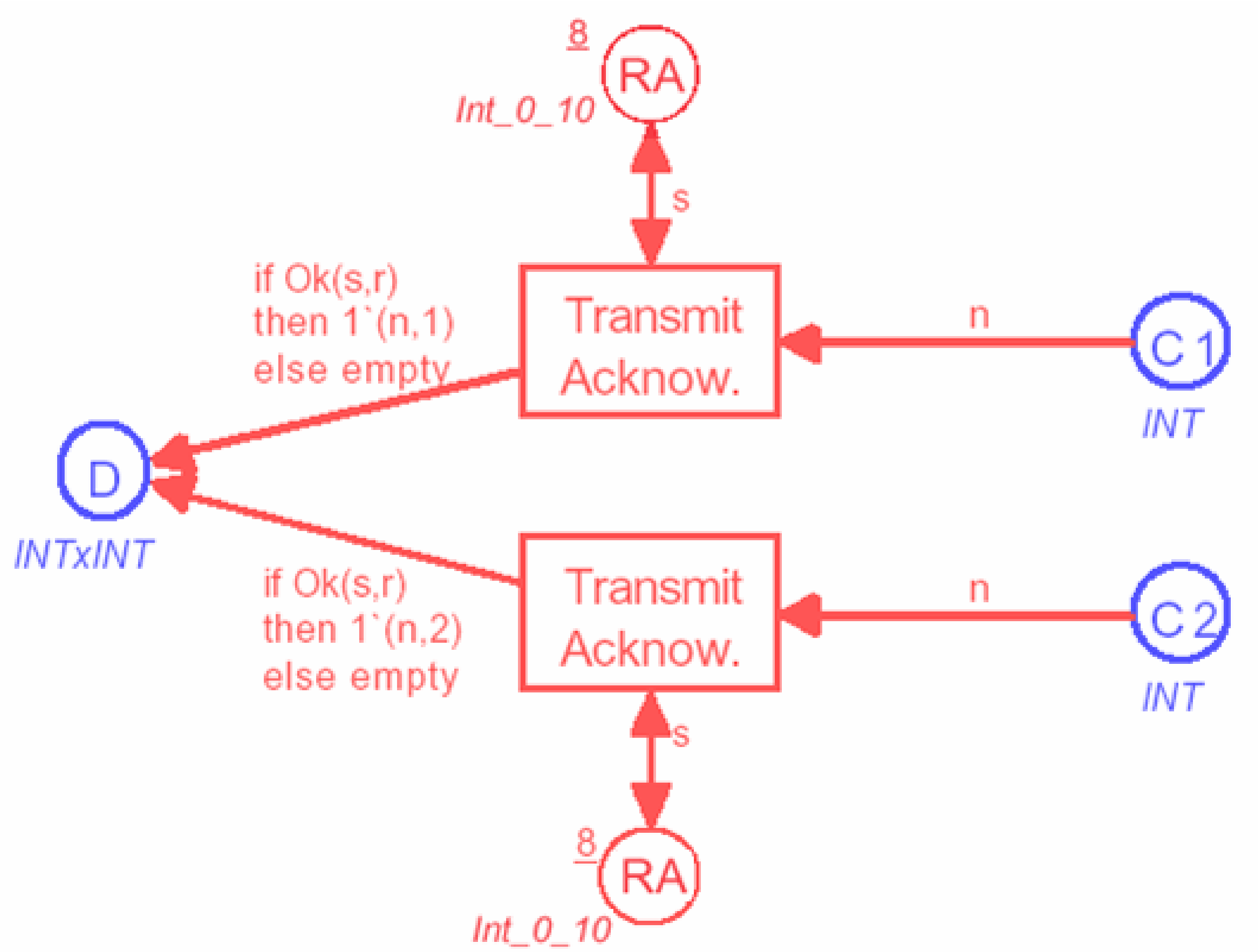


Transmit packets

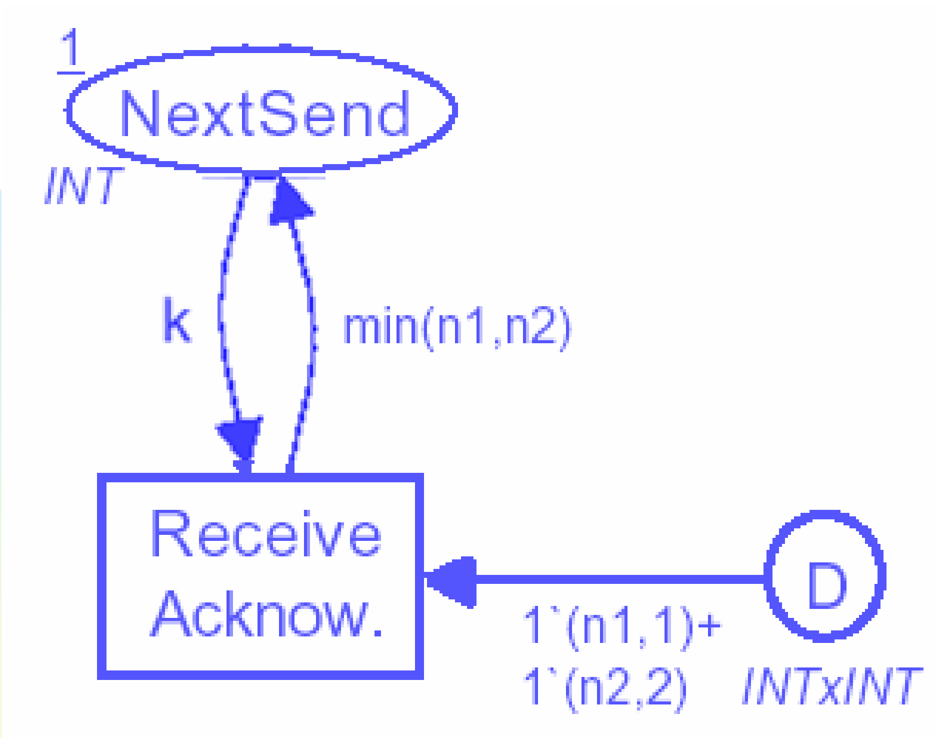


- Packets are *broadcasted* to the two receivers.
  - *Some* of the packets may *be lost.*

Transmit acknowledgments



Receive acknowledgments



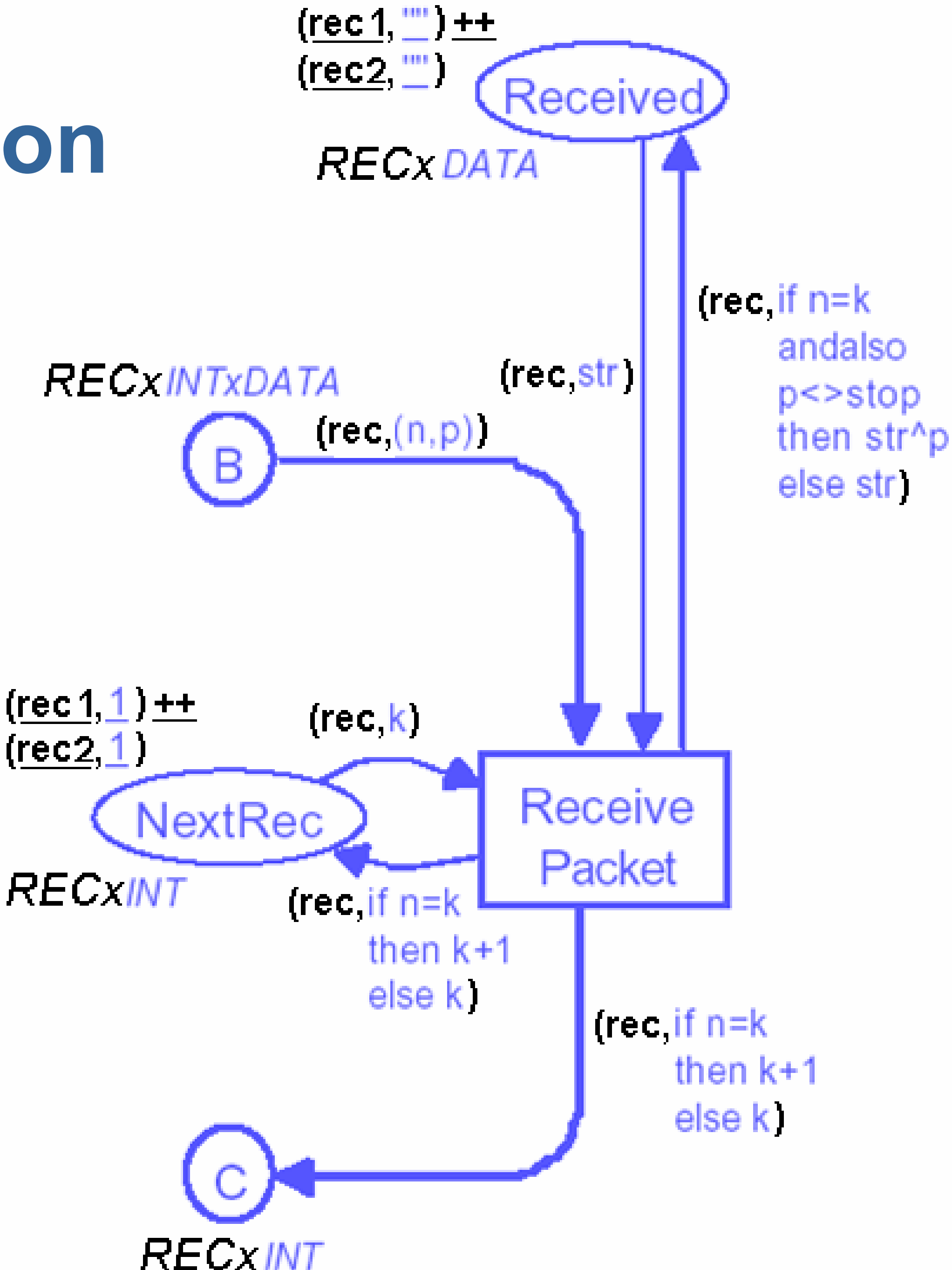
- The sender follows the *slowest* receiver.

Hierarchical descriptions

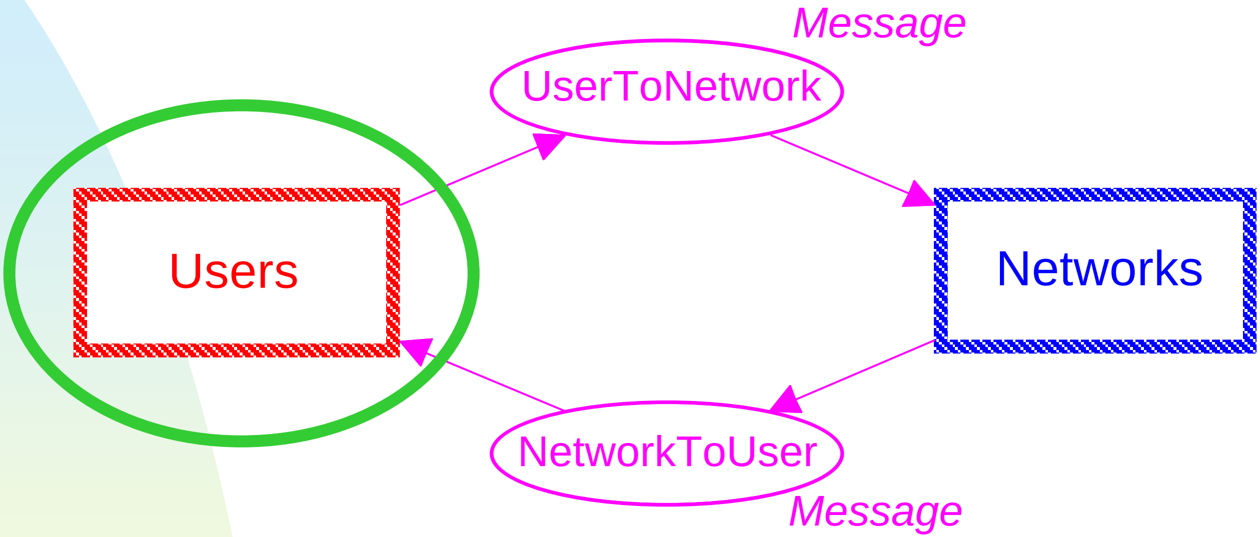
- We use *modules* to *structure* *large* and *complex* descriptions.
- Modules allow us to *hide details* that we do not want to consider at a certain *level of abstraction*.
- Modules have *well-defined interfaces*, consisting of *socket* and *port places*, through which the modules *exchange tokens*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

Another solution

- *Multiple receivers* may also be modelled by adding a *new component* to the *token colours.*
- Similar changes for *Transmit Packet* and *Transmit Acknowledgment.*

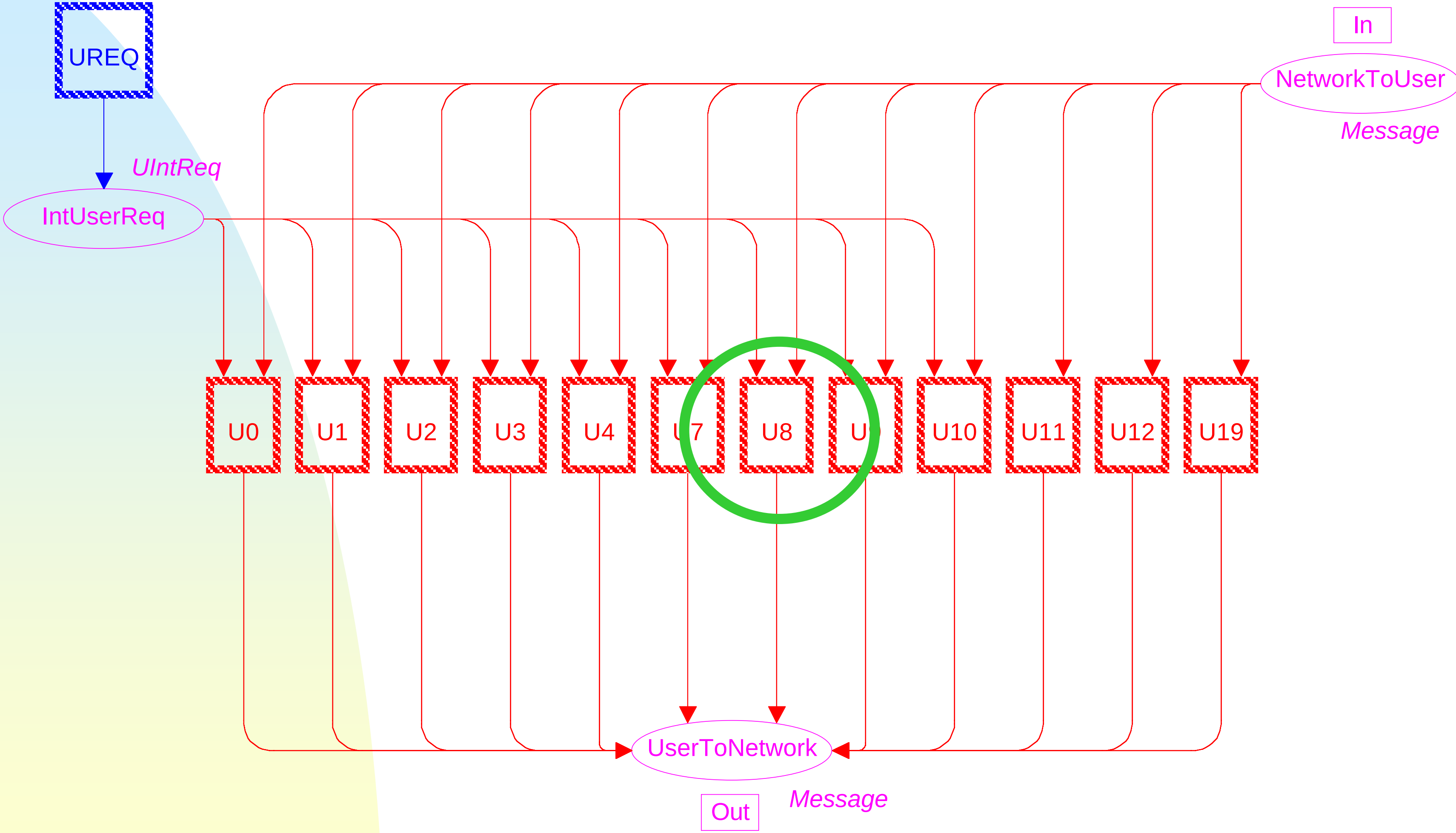


Protocol for ISDN network

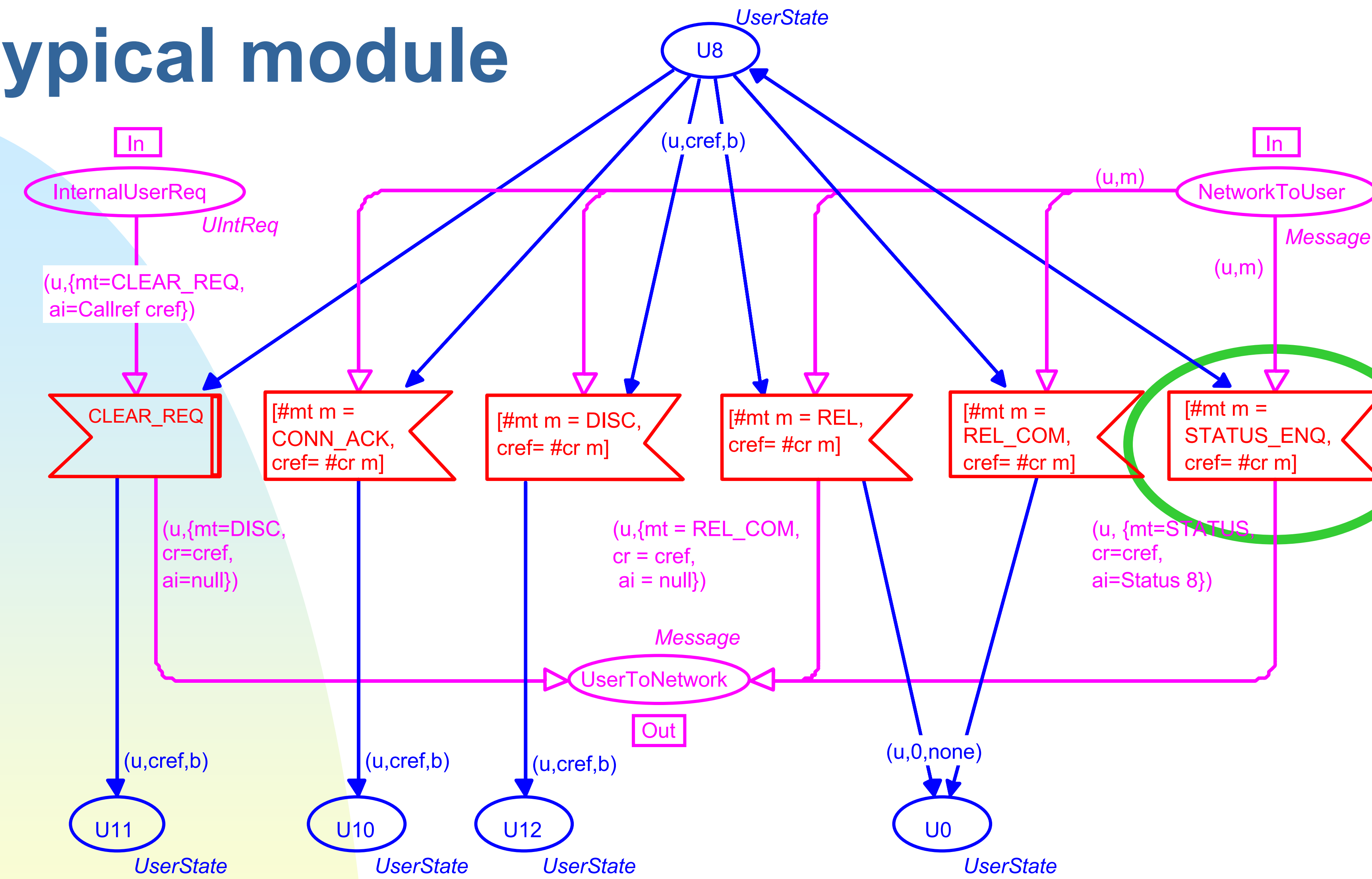


- Most *abstract view* of the system.

Overview of user site



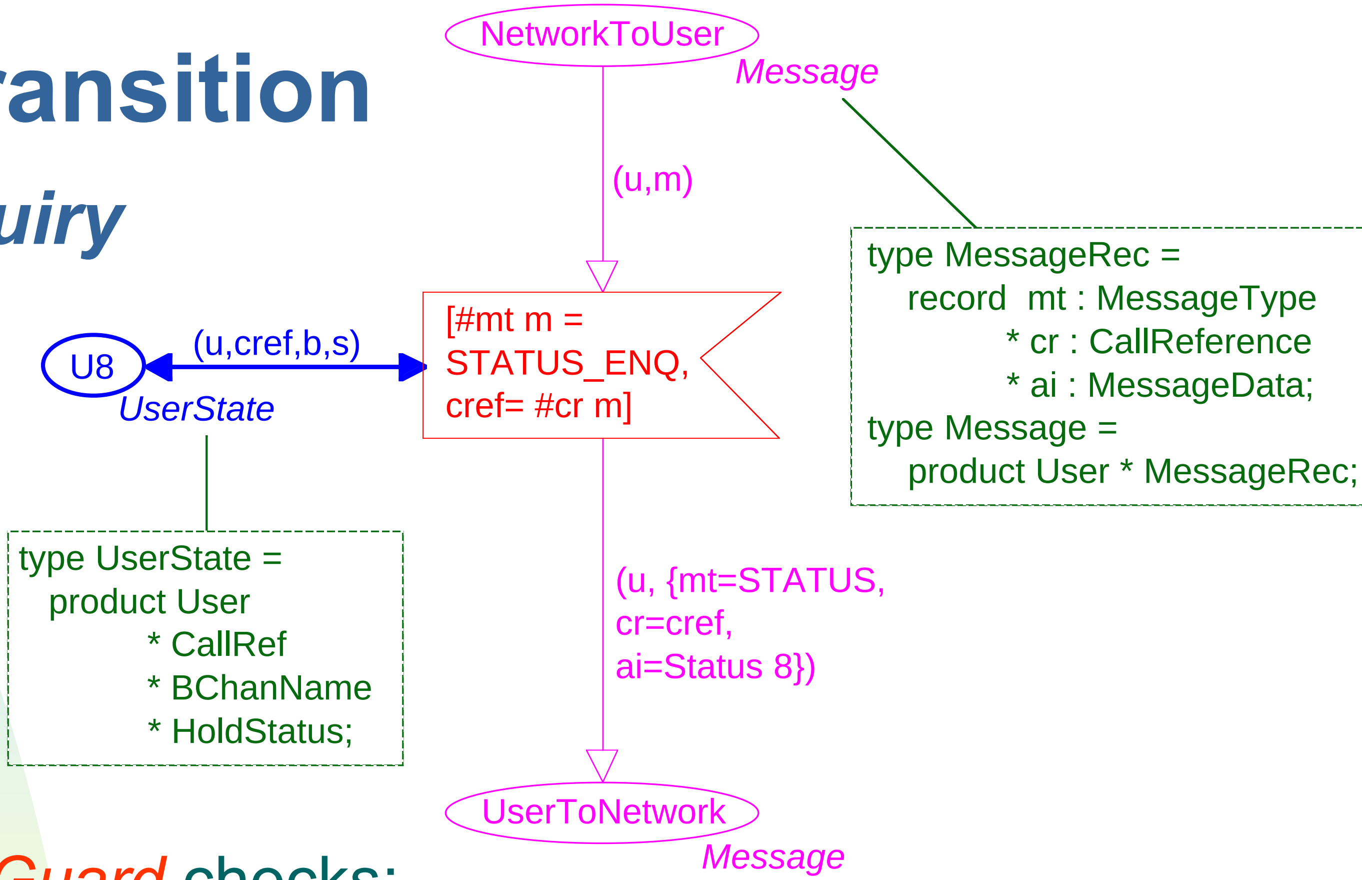
Typical module



- This *module* describes the *actions* that can happen when the *user site* is in state *U8.*
- The *node shapes* have a meaning in *SDL.*

Typical transition

***Status Enquiry* message received in state U8.**

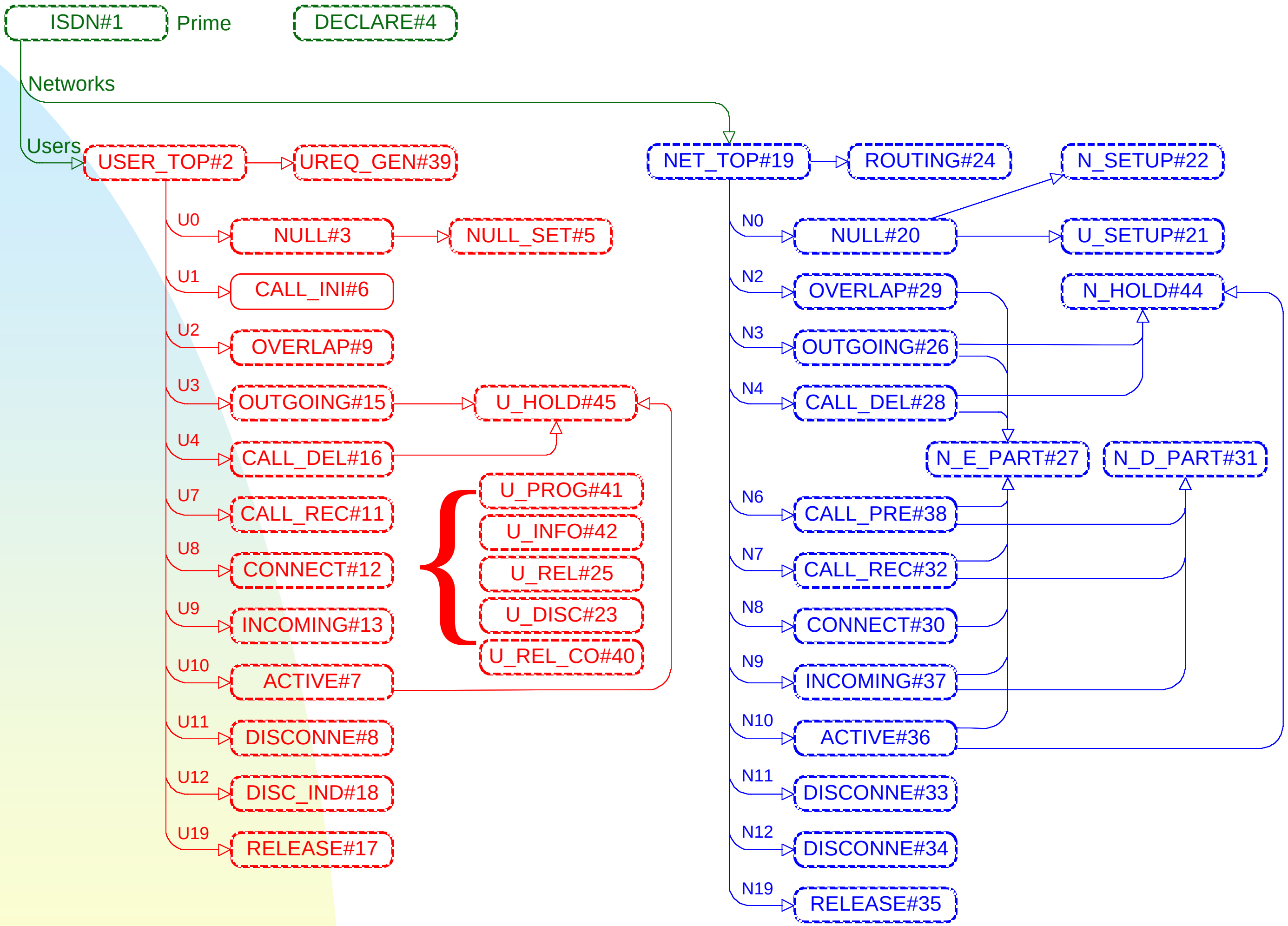


```
type MessageRec =
   record  mt : MessageType
            * cr : CallReference
            * ai : MessageData;
type Message =
   product User * MessageRec;
```

```
type UserState =
   product User
           * CallRef
           * BChanName
           * HoldStatus;
```

- *Guard* checks:
  - Message is a *Status Enquiry* message.
  - *Call Reference* is correct (i.e., matches the one in the *User State* token at place *U8*).
- A *Status message* is sent to the *network site*. It tells that the user site is in state *U8.*

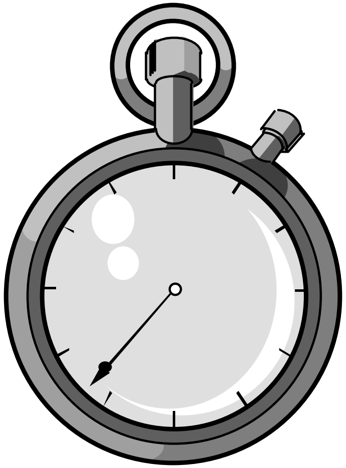
Some modules are used many times



- *43 modules* with more than *100 instances.*
- *Entire model* was made in only *3 man-weeks.*

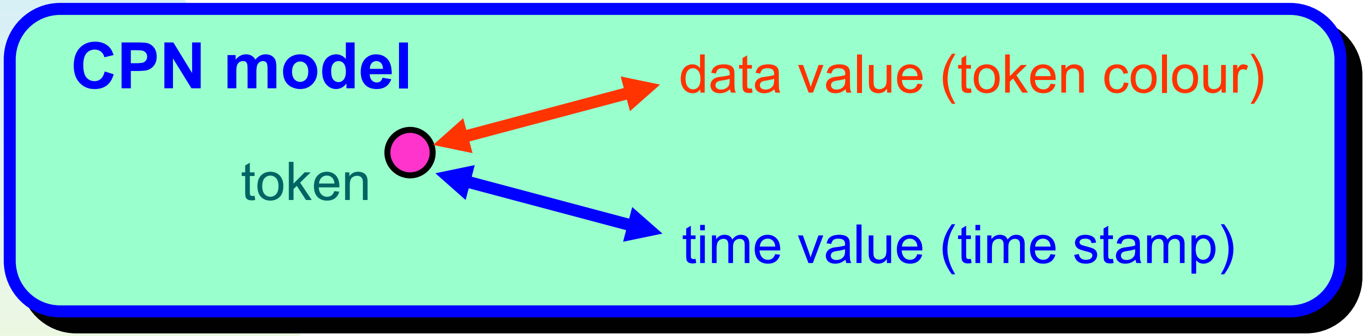
Time analysis

- CP-nets can be extended with a *time concept.* This means that the *same modelling language* can be used to investigate:
  - *Logical correctness.*
    Desired functionality, absence of deadlocks, etc.
  - *Performance*.
    How fast is the system and how many resources are used.



How to add time

- *Time* has been added to *Petri net models* in many different ways – typically by specifying *delays* on *places* or *transitions*.



CPN model

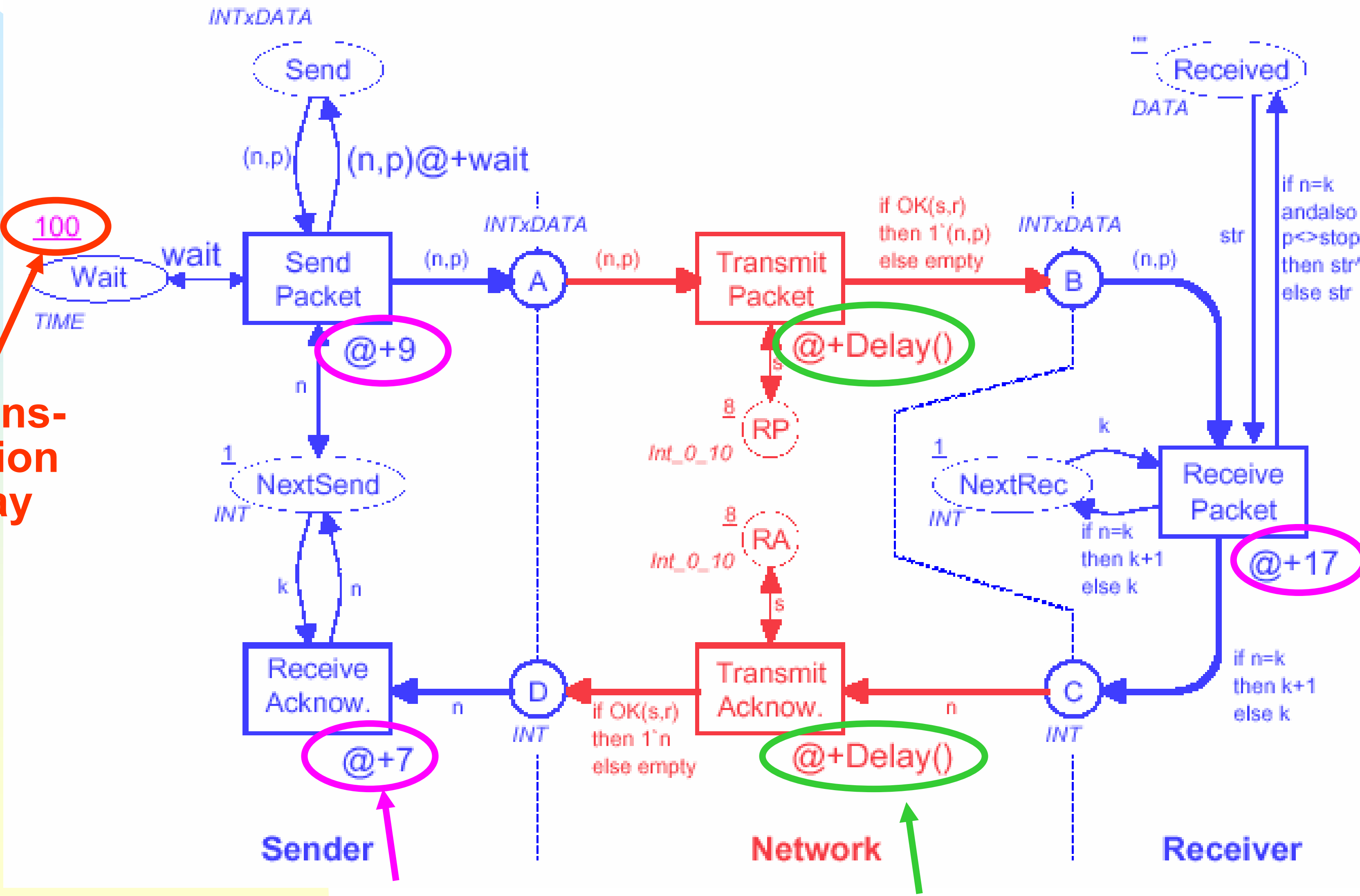
token

data value (token colour)

time value (time stamp)

- *Time stamp* determines *when* the token can be used, i.e., *consumed by a transition.*
  - *Delays* can be *fixed.*
  - Determined by an *arbitrary distribution*.

A timed CP-net for protocol



Retrans-mission delay

Fixed delay

Variable delay

Application areas

Protocols and Networks

- Intelligent Networks at Deutsche Telekom
- IEEE 802.6 Configuration Control at Telstra Research Labs
- Allocation Policies in the Fieldbus Protocol in Japan
- ISDN Services at Telstra Research Laboratories
- Protocol for an Audio/Video System at Bang & Olufsen
- TCP Protocols at Hewlett-Packard
- Local Area Network at University of Las Palmas
- UPC Algorithms in ATM Networks at University of Aarhus
- BRI Protocol in ISDN Networks
- Network Management System at RC International A/S
- Interprocess Communication in Pool IDA at King's College

Software

- Mobile Phones at Nokia
- Bank Transactions & Interconnect Fabric at Hewlett-Packard
- Mutual Exclusion Algorithm at University of Aarhus
- Distributed Program Execution at University of Aarhus
- Internet Cache at the Hungarian Academy of Science
- Electronic Funds Transfer in the US
- Document Storage System at Bull AG
- ADA Program at Draper Laboratories

Control of Systems

- Security and Access Control Systems at Dalcotech A/S
- Mechatronic Systems in Cars at Peugeot-Citroën in France
- European Train Control System in Germany
- Flowmeter System at Danfoss
- Traffic Signals in Brazil
- Chemical Production in Germany
- Model Train System at University of Kiel

Hardware

- Superscalar Processor Architectures at Univ. of Newcastle
- VLSI Chip in the US
- Arbiter Cascade at Meta Software Corp.

Military Systems

- Military Communications Gateway in Australia
- Influence Nets for the US Air Force
- Missile Simulator in Australia
- Naval Command and Control System in Canada

Other Systems

- Bank Courier Network at Shawmut National Coop.
- Nuclear Waste Management Programme in the US

Overview of talk

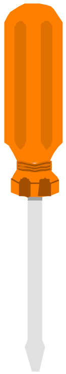
Modelling

- Basic language
  - syntax
  - semantics
- Extensions
  - modules
  - time
- **Tool support**
  - editing
  - simulation

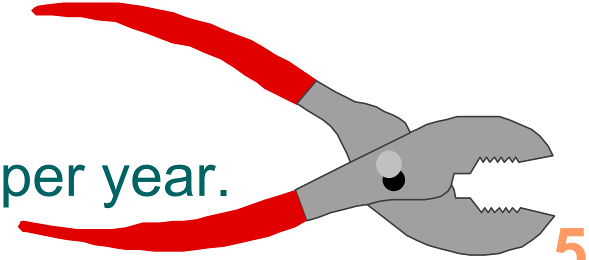
Analysis

- State spaces
  - full
  - symmetries
  - equivalence classes
  - sweep-line
- Place invariants
  - check of invariants
  - use of invariants

Computer tools



- *Design/CPN* was developed in the late 80'ies and early 90'ies.
  - Until recently, it was the *most widely used* Petri net package.
  - Used by *1000 different organisations* in more than *60 countries* – including *200 commercial companies.*
- *CPN Tools* is the *next generation* of tool support for Coloured Petri Nets.
  - It has now *replaced Design/CPN* with *1250 users* in more than *75 countries.*
  - Development *started in 1999* and a total of *25 man-years* have been used.
  - Development *continues* with an expected effort of *5 man-years* per year.



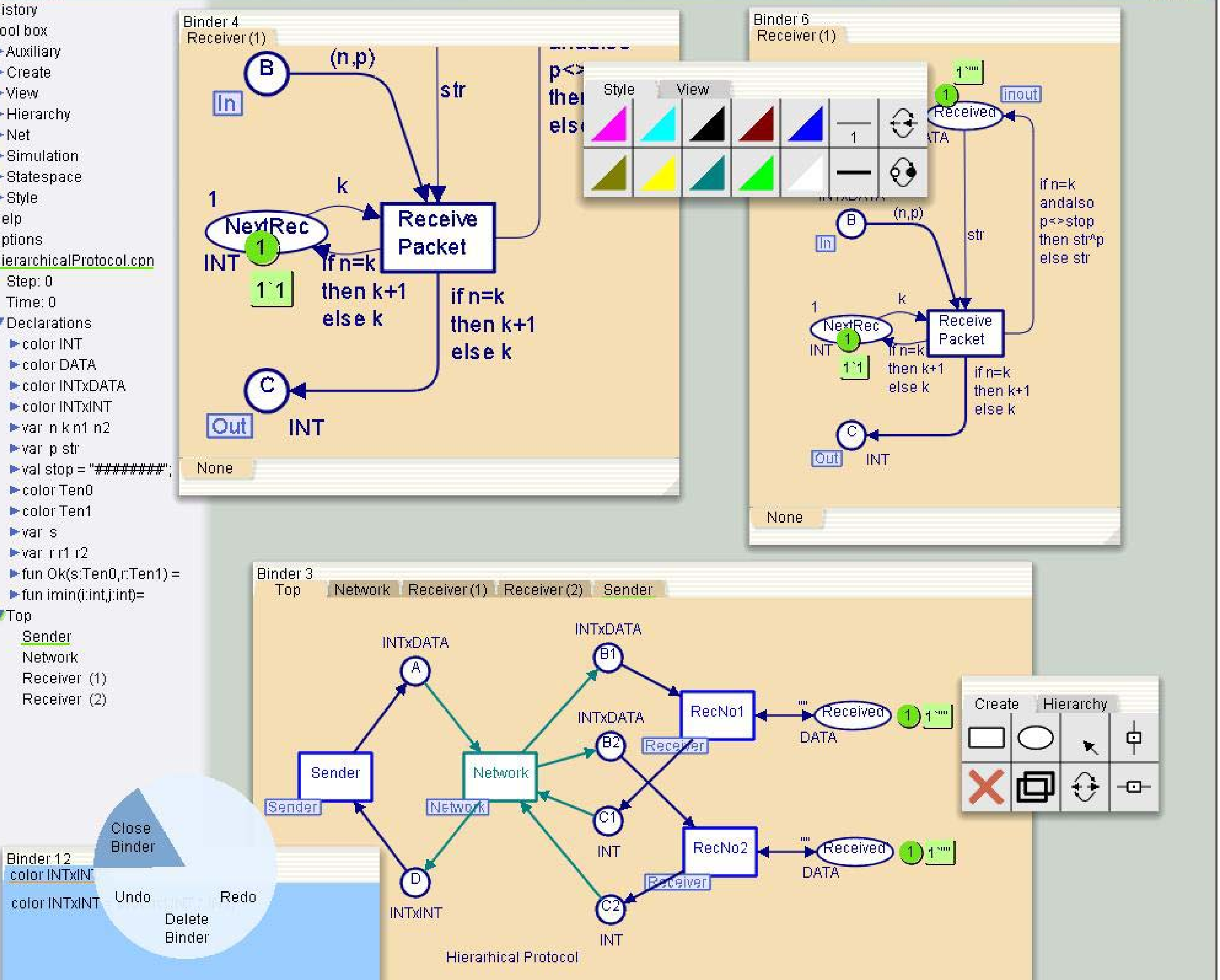
CPN Tools and Design/CPN

The *functionality* of the two tools is the *same:*

- *Editing* and *syntax check* of CP-nets.
- *Interactive* and *automatic simulation*.
- *Construction* and *analysis* of *state spaces.*
- *Communication* with other tools.
- Simulation based *performance analysis.*
- *Graphical animation* of simulation results.

What is new in CPN Tools?

- *Windows XP.* Later versions will also support *Linux.*
- *On-the-fly, incremental syntax check.*
- *Much more efficient simulation engine* in particular for:
  - Models with *many tokens.*
  - *Timed* models.
- *New user interface* with a number of state-of-the-art interaction mechanisms:
  - Possible to have a *mouse in each hand.*
  - Tool glasses, floating palettes and circular marking menus.

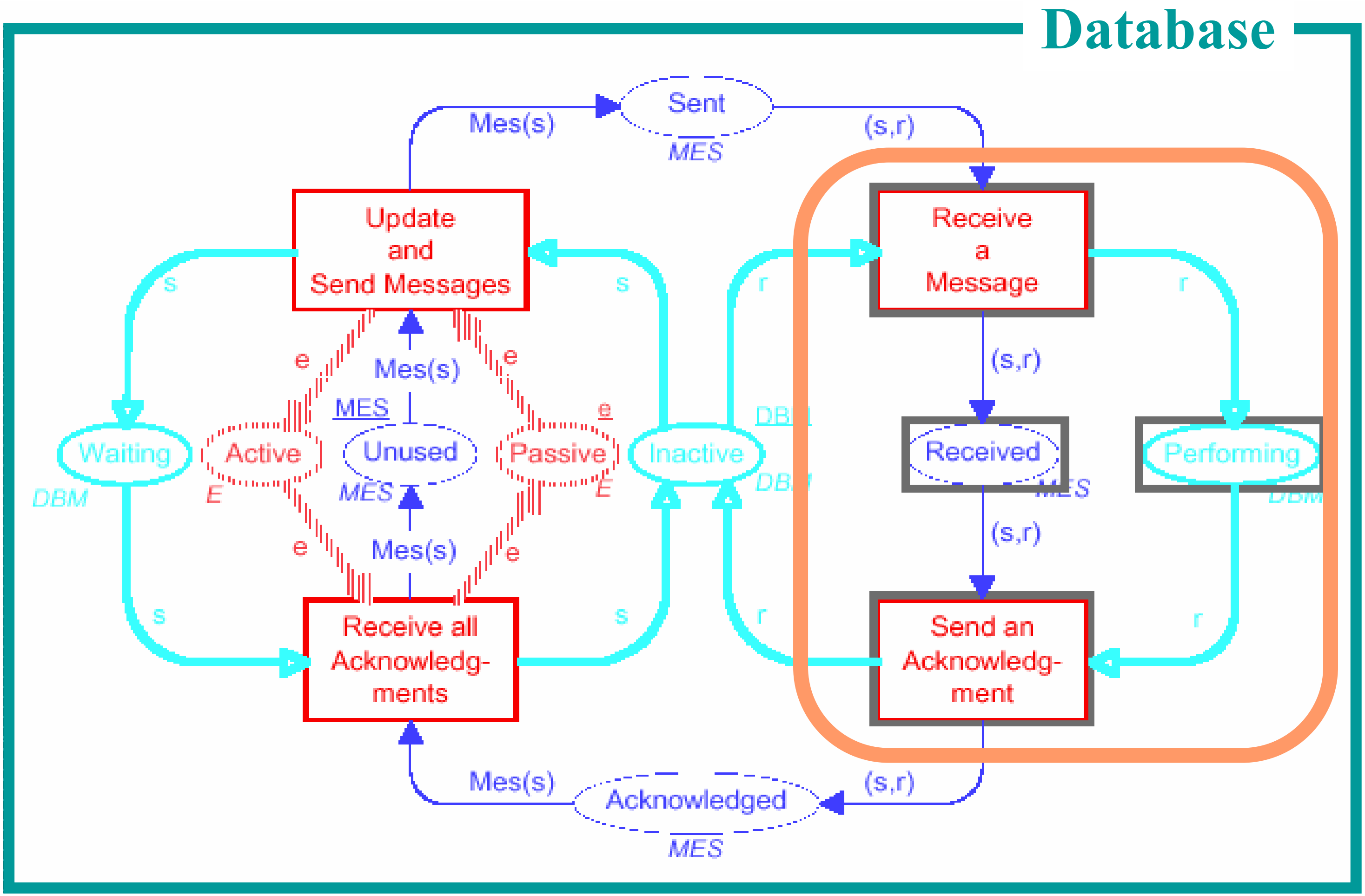


istory
ool box
Auxiliary
Create
View
Hierarchy
Net
Simulation
Statespace
Style
elp
ptions
ierarchicalProtocol.cpn
Step: 0
Time: 0
Declarations
color INT
color DATA
color INTxDATA
color INTxINT
var n k n1 n2
var p str
val stop = "########";
color Ten0
color Ten1
var s
var r r1 r2
fun Ok(s:Ten0,r:Ten1) =
fun imin(i:int,j:int)=
Top
Sender
Network
Receiver (1)
Receiver (2)
Binder 4
Receiver (1)
B
In
(n,p)
str
andalso
p<>
then
else
1
NextRec
1
INT
1`1
k
Receive Packet
if n=k then k+1 else k
if n=k then k+1 else k
C
Out
INT
None
Style
View
1
Binder 6
Receiver (1)
1`""
1
Received
inout
DATA
INTxDATA
B
In
(n,p)
str
if n=k andalso p<>stop then str^p else str
1
k
NextRec
1
INT
1`1
Receive Packet
if n=k then k+1 else k
if n=k then k+1 else k
C
Out
INT
None
Binder 3
Top
Network
Receiver (1)
Receiver (2)
Sender
INTxDATA
A
INTxDATA
B1
RecNo1
""
Received
1
1`""
DATA
INTxDATA
B2
Receiver
Sender
Sender
Network
Network
C1
INT
RecNo2
""
Received
1
1`""
DATA
Receiver
D
INTxINT
C2
INT
Hierarhical Protocol
Create
Hierarchy
Binder 12
color INTxINT
color INTxINT
Close Binder
Undo
Redo
Delete Binder

Standard ML

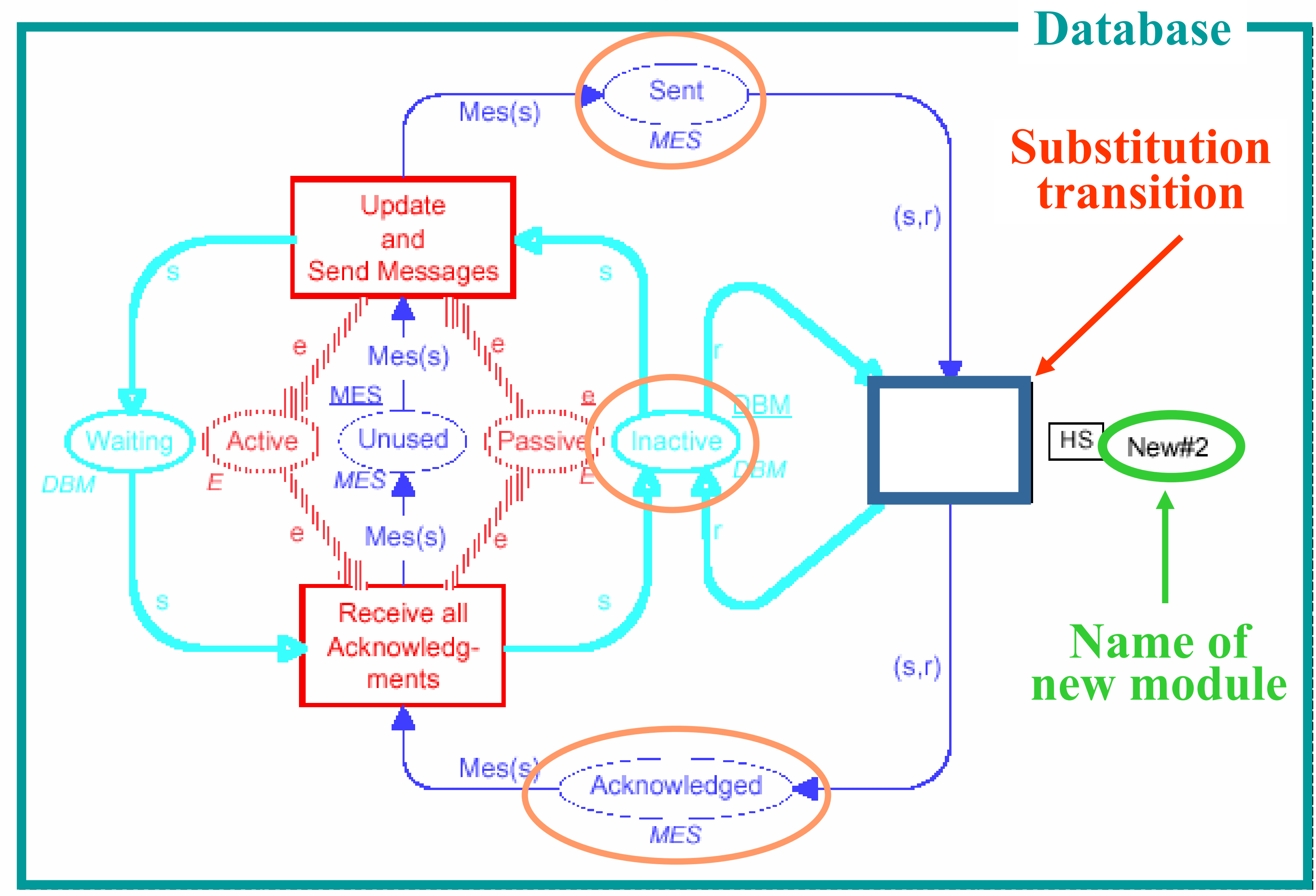
- Types, arc expressions and guards are specified in *Standard ML*, which is a strongly typed, functional programming language developed by *Robin Milner*.
- *Data types* can be:
  - *Atomic* (integers, strings, booleans and enumerations).
  - *Structured* (products, records, unions, lists, and subsets).
- Arbitrary complex *functions* and *operations* can be defined (e.g., using polymorphism).
- Standard ML is well-known, well-tested and very general. Several *text books* are available.

Support for hierarchical models



- We want to move the ***selected part*** to a *new module*
- This is done by a *single operation*.

Abstract view



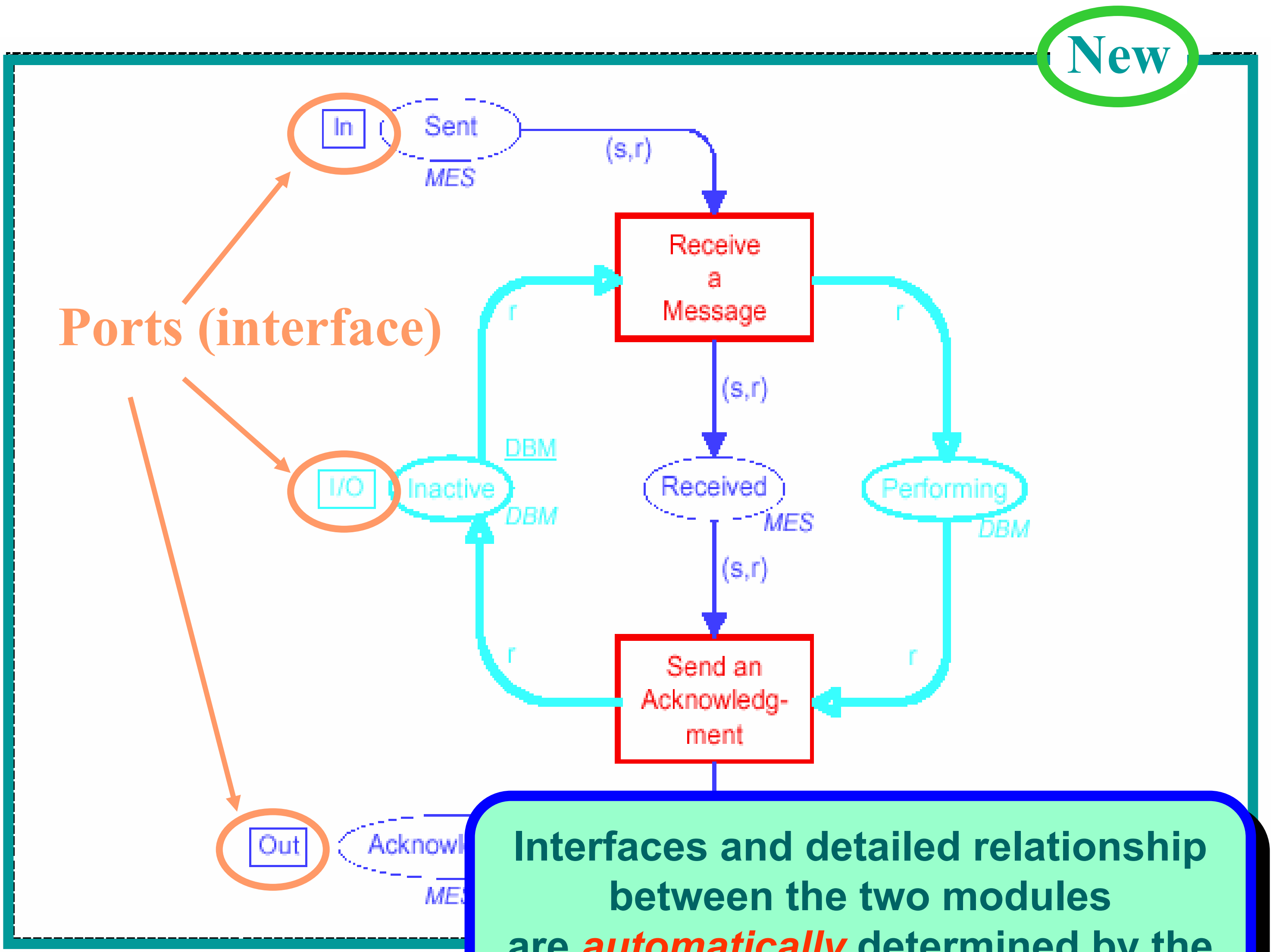
Sockets (interface)

Substitution transition

Name of new module

Detailed view

Name of new module



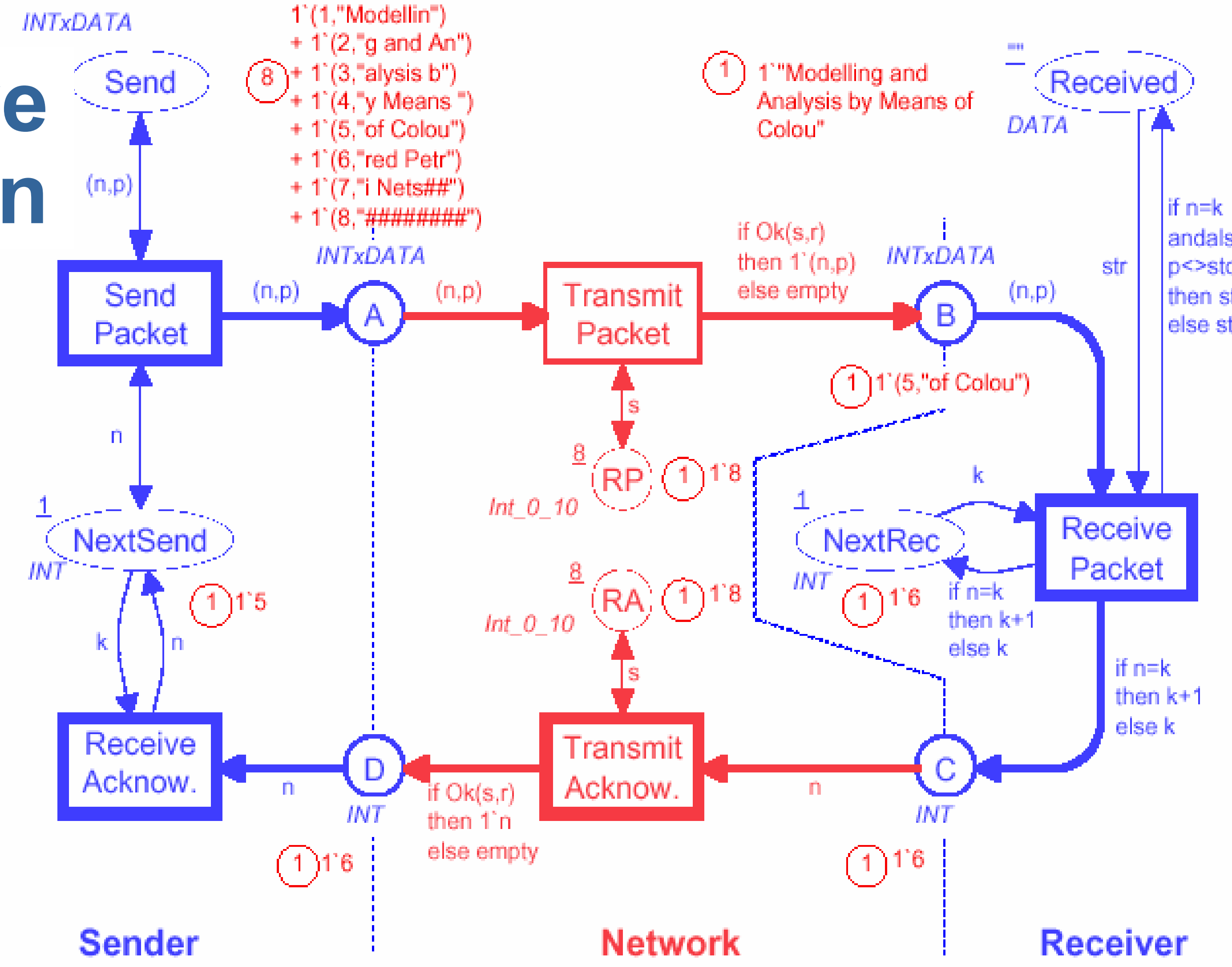
Ports (interface)

Interfaces and detailed relationship between the two modules are *automatically* determined by the CPN editor.

Simulation of CP-nets

- When a *syntactical correct* CPN diagram has been constructed, the CPN tool *generates* the necessary *code* to *perform simulations.*
  - Calculates whether the individual transitions and bindings are *enabled.*
  - Calculates the *effect* of occurring transitions and bindings.
- The *syntax check* and *code generation* are *incremental*. Hence it is fast to make small changes to the CPN diagram.
- We distinguish between *two kinds of simulations:*
  - In an *interactive* simulation the user is in control,but most of the work is done by the system.
  - In an *automatic* simulation the system does all the work.

Interactive simulation



- *Simulation results* are shown directly on the CP-net.
  - *Transitions* are chosen by the *user or the simulator.*
  - User can *observe* all details and set *breakpoints.*

Automatic simulation

- The user does *not* intend to follow the simulation:
  - Simulation can be *very fast* - several thousand steps per second.
  - User specifies some *stop criteria*, which determine the duration of the simulation.
  - When the simulation stops the *graphics* of the CP-net is *updated.*
  - Then the user can *inspect* all details of the graphics, e.g., the *enabling* and the *marking.*
- Automatic simulations can be *mixed* with interactive simulations.
- To find out what happens *during* an automatic simulation the user has a number of choices.

Simulation report

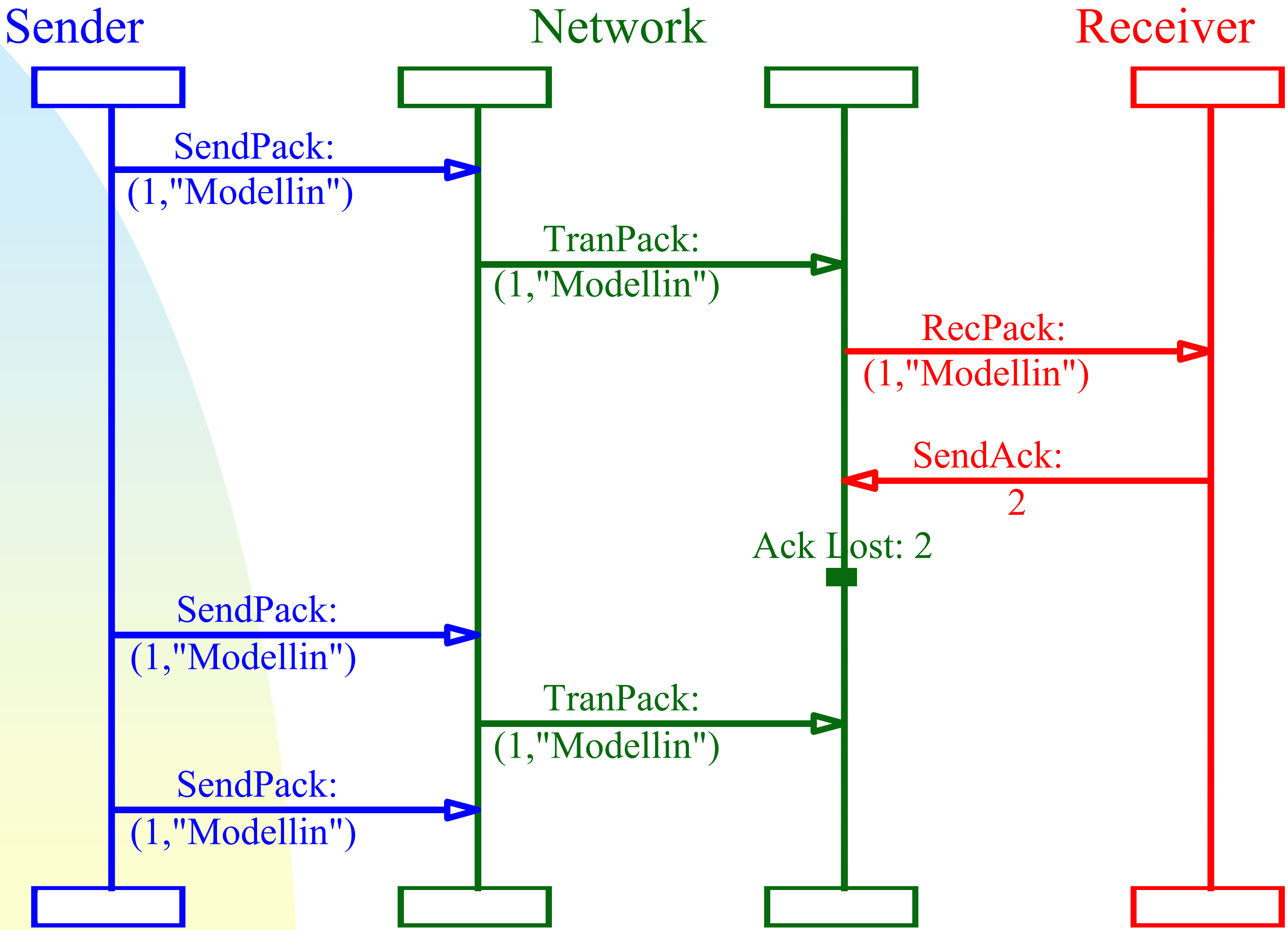
```
1 SendPack@(1:Top#1){n=1,p="Modellin"}
2 TranPack@(1:Top#1){n=1,p="Modellin",r=6,s=8
3 SendPack@(1:Top#1){n=1,p="Modellin"}
4 TranPack@(1:Top#1){n=1,p="Modellin",r=3,s=8
5 RecPack@(1:Top#1) {k=1,n=1,p="Modellin",str
6 SendPack@(1:Top#1){n=1,p="Modellin"}
```

Transition

Module

Binding

Message sequence chart



Sender

Network

Receiver

SendPack: (1,"Modellin")

TranPack: (1,"Modellin")

RecPack: (1,"Modellin")

SendAck: 2

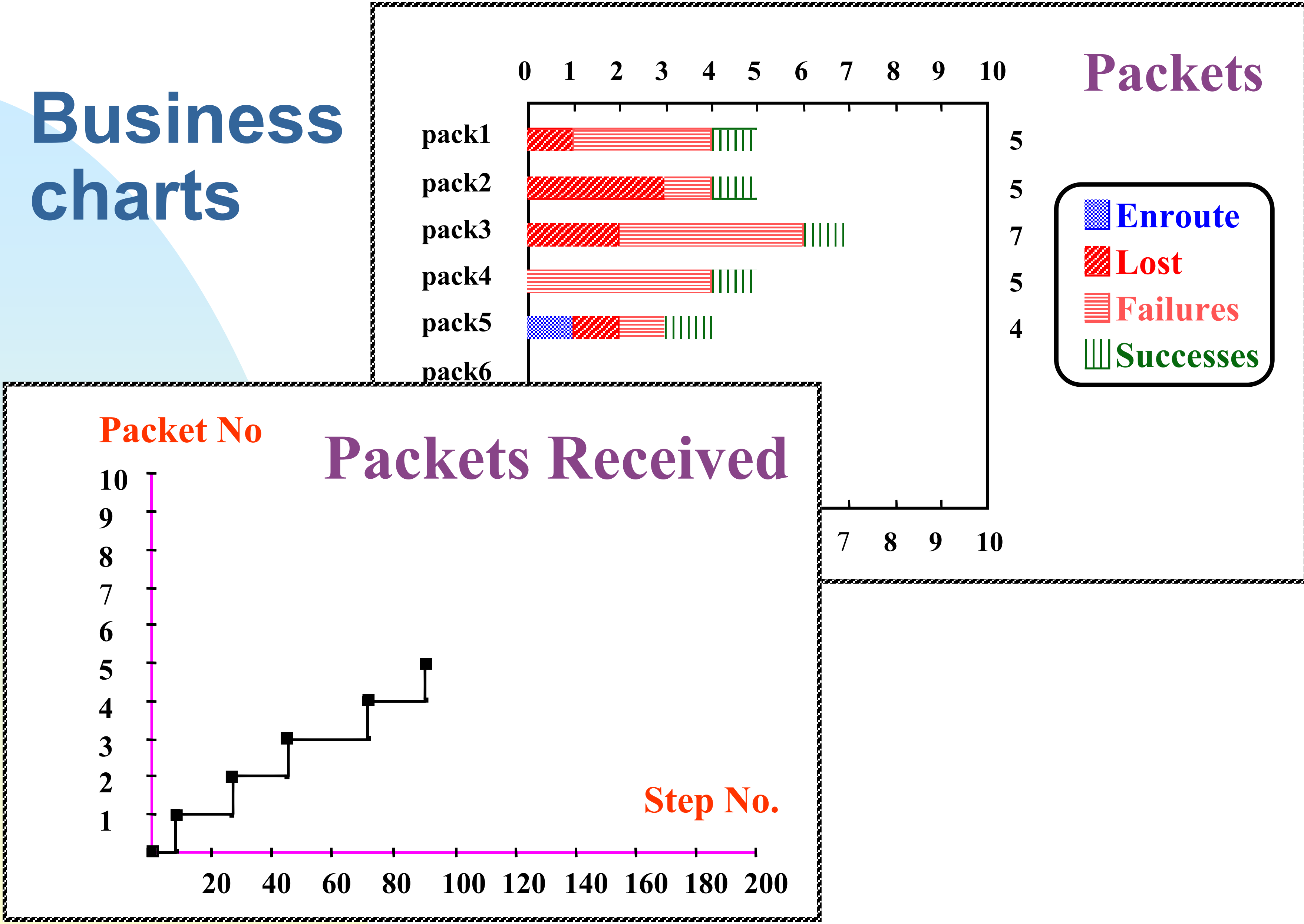
Ack Lost: 2

SendPack: (1,"Modellin")

TranPack: (1,"Modellin")

SendPack: (1,"Modellin")

Business charts



Automatic code generation

- CPN models are often used to *specify* and *validate* new software.
- It is also possible to *implement* the software by *automatic code generation.*
  - This method has been applied to develop a system for *access control* to buildings.
  - The source code for the final implementation was generated *automatically* from the CPN specification - by extracting parts of the *Standard ML code* used by the CPN simulator.
  - The approach is only adequate for systems that are *not time critical* and systems that are produced in *small numbers.*

Overview of talk

Modelling

- Basic language
  - syntax
  - semantics
- Extensions
  - modules
  - time
- Tool support
  - editing
  - simulation

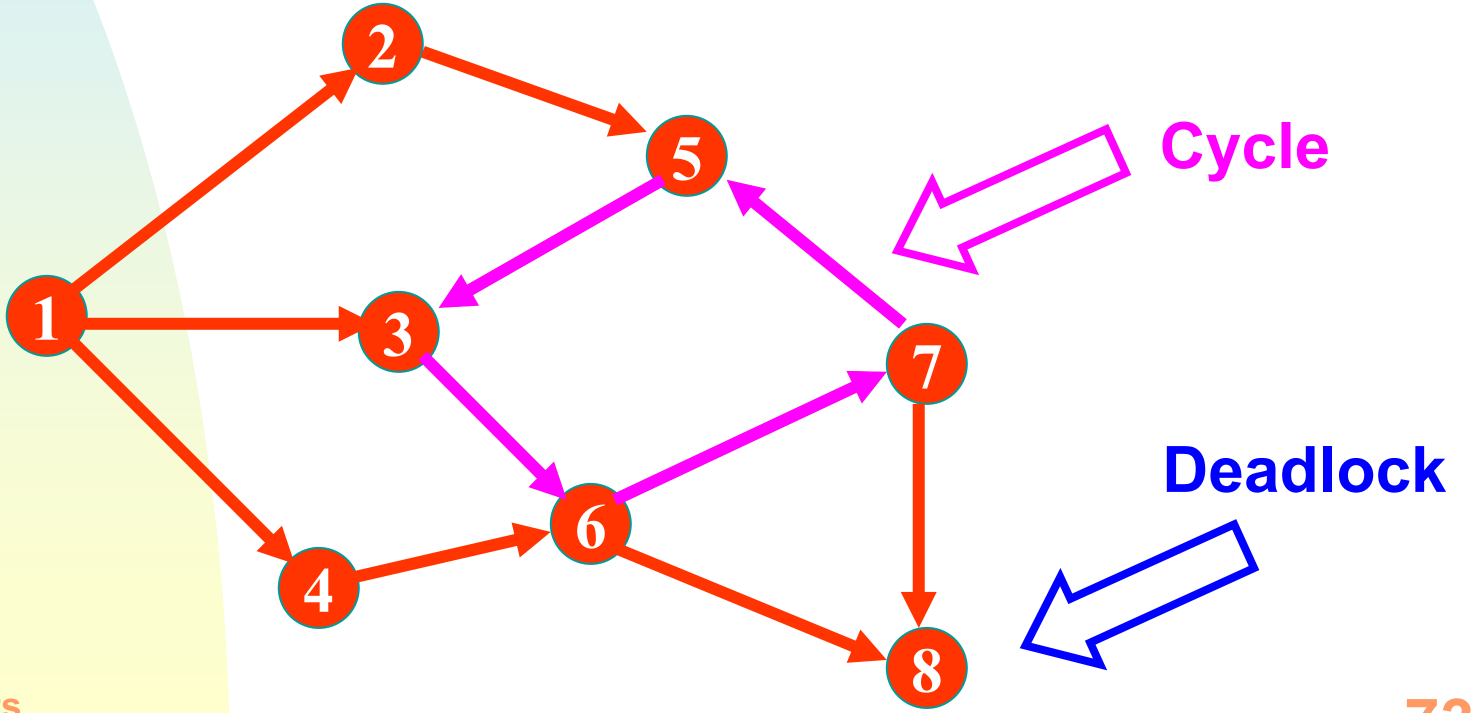
Analysis

- **State spaces**
  - full
  - symmetries
  - equivalence classes
  - sweep-line
- Place invariants
  - check of invariants
  - use of invariants

State spaces

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

transition + binding



Cycle

Deadlock

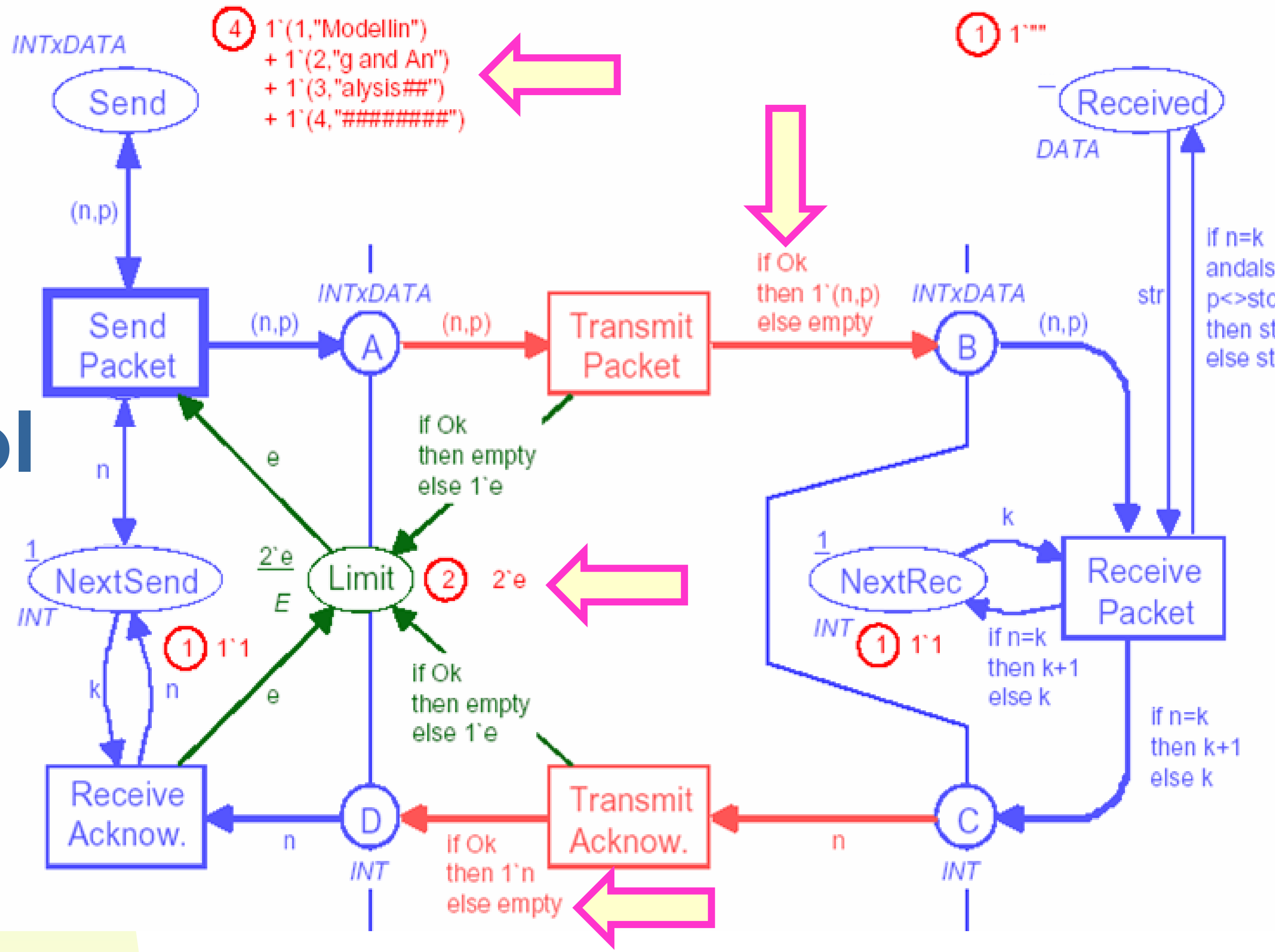
State space tool

- 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.
- *Generation* is totally *automatic* while *analysis* is *automatic* or *semi-automatic* (based on queries from the user).

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 space for protocol



- To obtain a *finite* state space, we:
  - Only have *4 packets.*
  - *Limit* the number of tokens on A, B, C, and D.
  - *Binary choice* between success and failure.

State space report for protocol

```
Occurrence Graph Statistics
   Nodes:      428
   Arcs:       1130
   Secs:       0
   Status:     Full

Scc Graph Statistics
   Nodes:      182
   Arcs:       673
   Secs:       0
```

Integer bounds

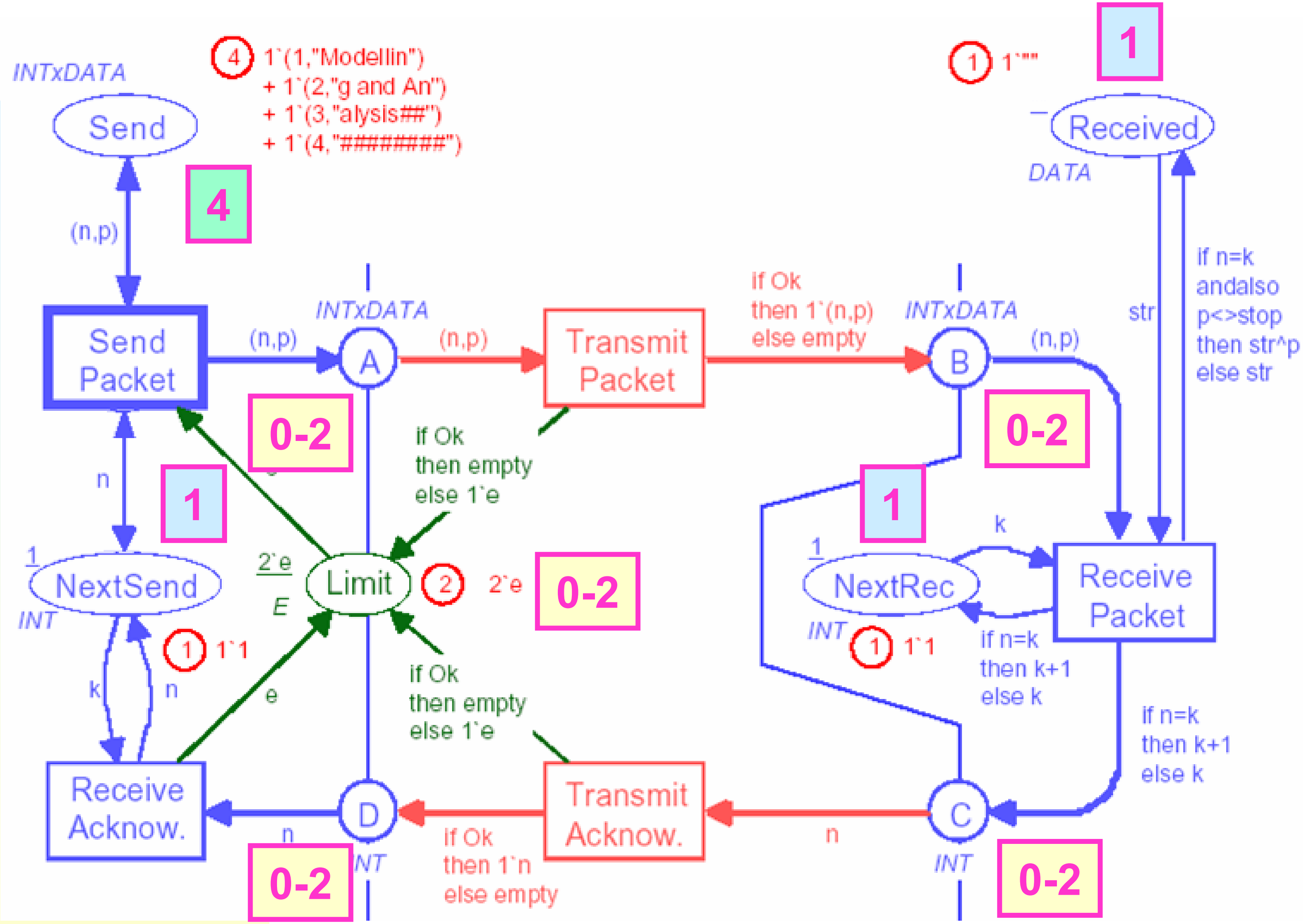
A, B, C, D, Limit: 0-2

NextSend, NextRec, Received: 1

Send: 4

- *Integer bounds* tell the *maximal* and *minimal number of tokens* on the individual places.

Integer bounds



Upper multi-set bounds

```
A, B:          2`(1,"Modellin") + 2`(2,"g and An") +
               2`(3,"alysis##") + 2`(4,"#########")

C, D:          2`2 + 2`3 + 2`4 + 2`5

Limit:         2`e

NextSend,
NextRec:       1`1 + 1`2 + 1`3 + 1`4 + 1`5

Received:      1`"" + 1`" Modellin" +
               1`"Modelling and An" +
               1`"Modelling and Analysis##"

Send:          1`(1,"Modellin") + 1`(2,"g and An") +
               1`(3,"alysis##") + 1`(4,"#########")
```

Home and liveness properties

Home Properties

Home Markings: [235]

Liveness Properties

Dead Markings: [235]

Live Transitions: None

NextSend = 5
NextRec = 5
Received = "Modelling and Analysis##"

235

Marking no. 235 is the *desired final marking* where all packets have been *received in correct order.*

Investigation of dead marking

- Marking 235 is the *only dead marking.*
    - This implies that the protocol is *partially correct* (if execution stops it stops in the desired final marking).
- Marking 235 is a *home marking.*
    - This implies that we *always have a chance to finish correctly* (it is impossible to reach a state from which we cannot reach the desired final marking).

Fairness properties

Send Packet:	**Impartial**
Transmit Packet:	**Impartial**
Receive Packet:	**No Fairness**
Transmit Acknow:	**No Fairness**
Receive Acknow:	**No Fairness**

◆ *Fairness properties* tell *how often* the individual transitions *occur.*

Investigation of shortest path

- We want to find one of the *shortest paths* from the *initial marking* to the *dead marking*.

```
val path =
NodesInPath(1,235);
```

```
> val path =
[1,2,3,4,6,8,10,15,20,27,50,
64,80,102,133,164,179,192,
201,215,235] : Node list
```

```
Length(path);
```

```
> 20 : int
```

Query

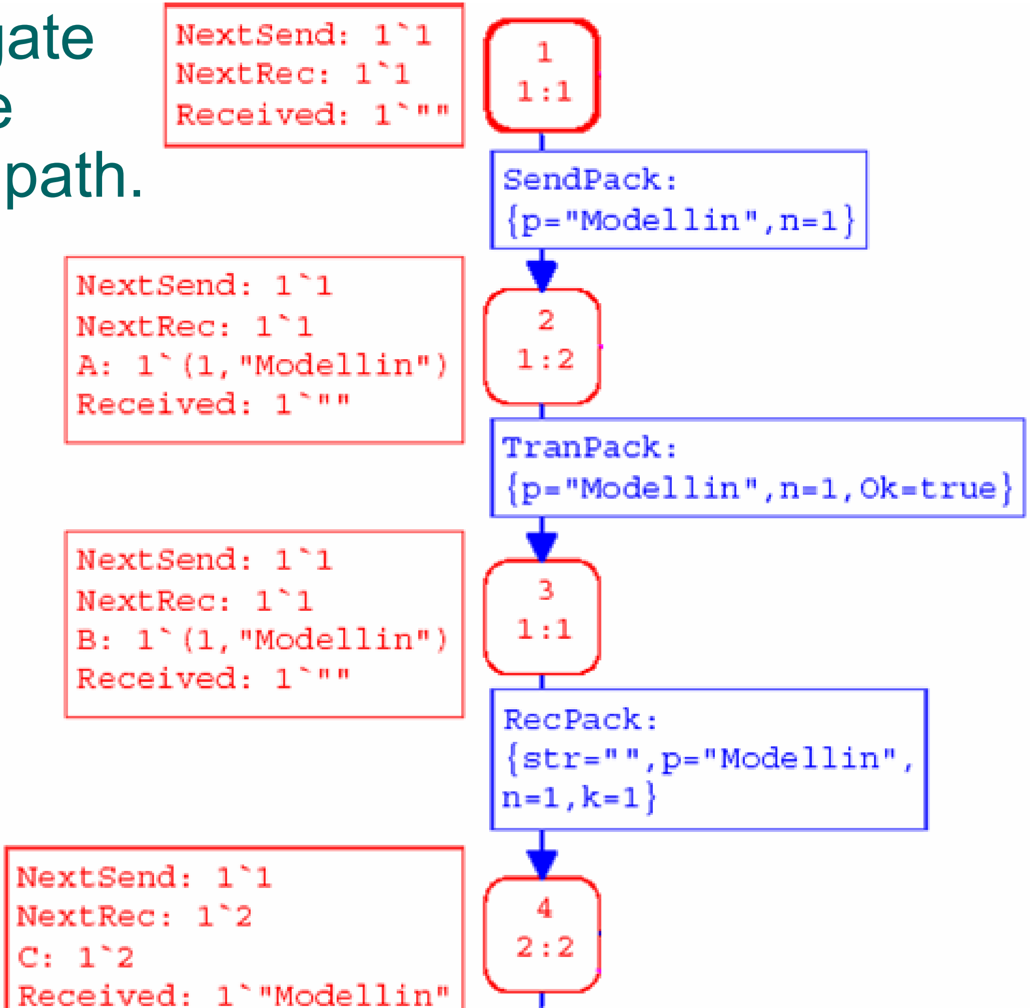
Answer

Drawing of shortest path

DisplayNodePath [1,2,3,4,6,8];

> () : unit

- We want to investigate the beginning of the calculated shortest path.



```
NextSend: 1`1
NextRec: 1`1
Received: 1`""
```

```
SendPack:
{p="Modellin",n=1}
```

```
NextSend: 1`1
NextRec: 1`1
A: 1`(1,"Modellin")
Received: 1`""
```

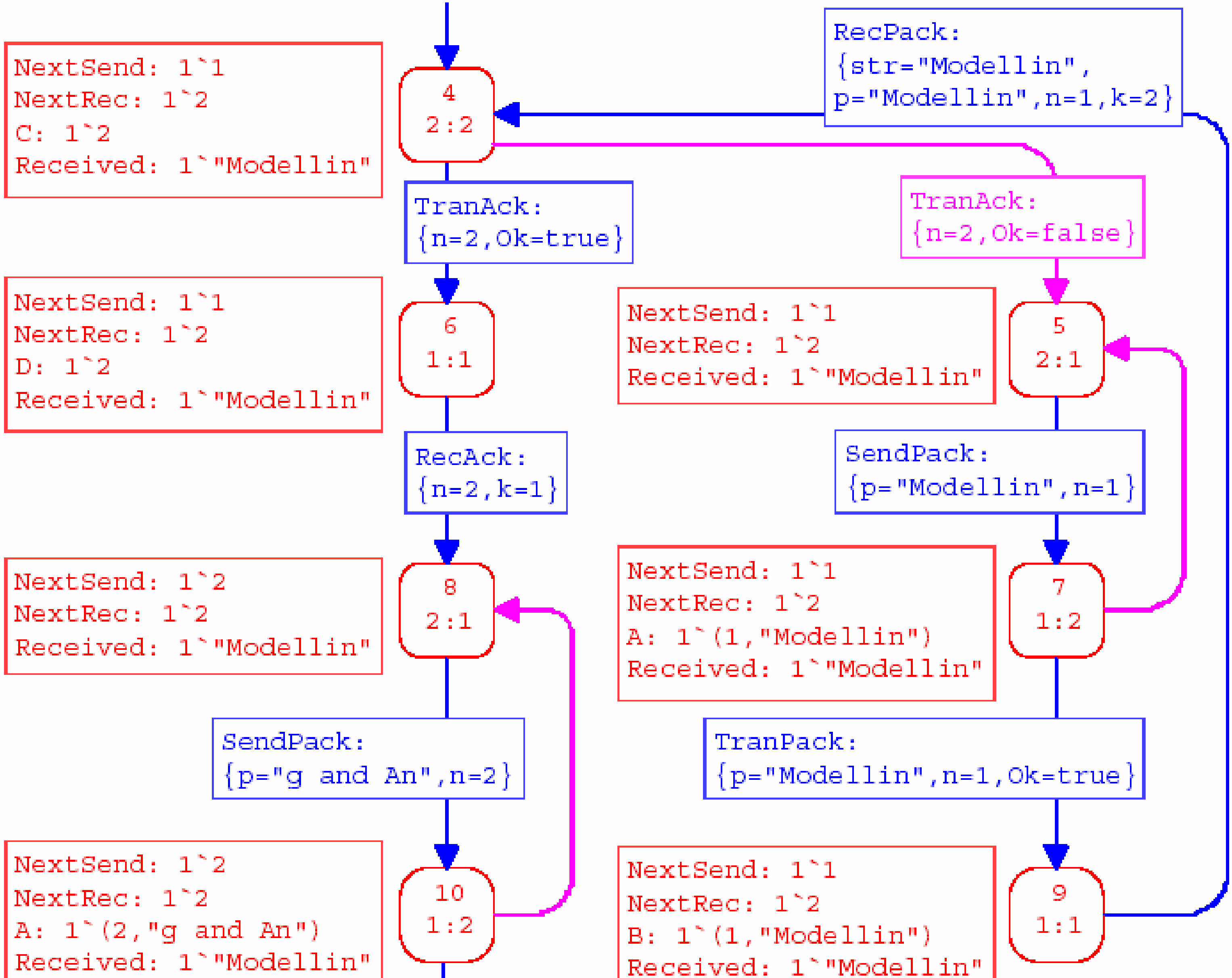
```
TranPack:
{p="Modellin",n=1,Ok=true}
```

```
NextSend: 1`1
NextRec: 1`1
B: 1`(1,"Modellin")
Received: 1`""
```

```
RecPack:
{str="",p="Modellin",
n=1,k=1}
```

```
NextSend: 1`1
NextRec: 1`2
C: 1`2
Received: 1`"Modellin"
```

Draw more complex subgraph



RecPack:
{str="Modellin",
p="Modellin",n=1,k=2}

NextSend: 1`1
NextRec: 1`2
C: 1`2
Received: 1`"Modellin"

4
2:2

TranAck:
{n=2,Ok=true}

TranAck:
{n=2,Ok=false}

NextSend: 1`1
NextRec: 1`2
D: 1`2
Received: 1`"Modellin"

6
1:1

NextSend: 1`1
NextRec: 1`2
Received: 1`"Modellin"

5
2:1

RecAck:
{n=2,k=1}

SendPack:
{p="Modellin",n=1}

NextSend: 1`2
NextRec: 1`2
Received: 1`"Modellin"

8
2:1

NextSend: 1`1
NextRec: 1`2
A: 1`(1,"Modellin")
Received: 1`"Modellin"

7
1:2

SendPack:
{p="g and An",n=2}

TranPack:
{p="Modellin",n=1,Ok=true}

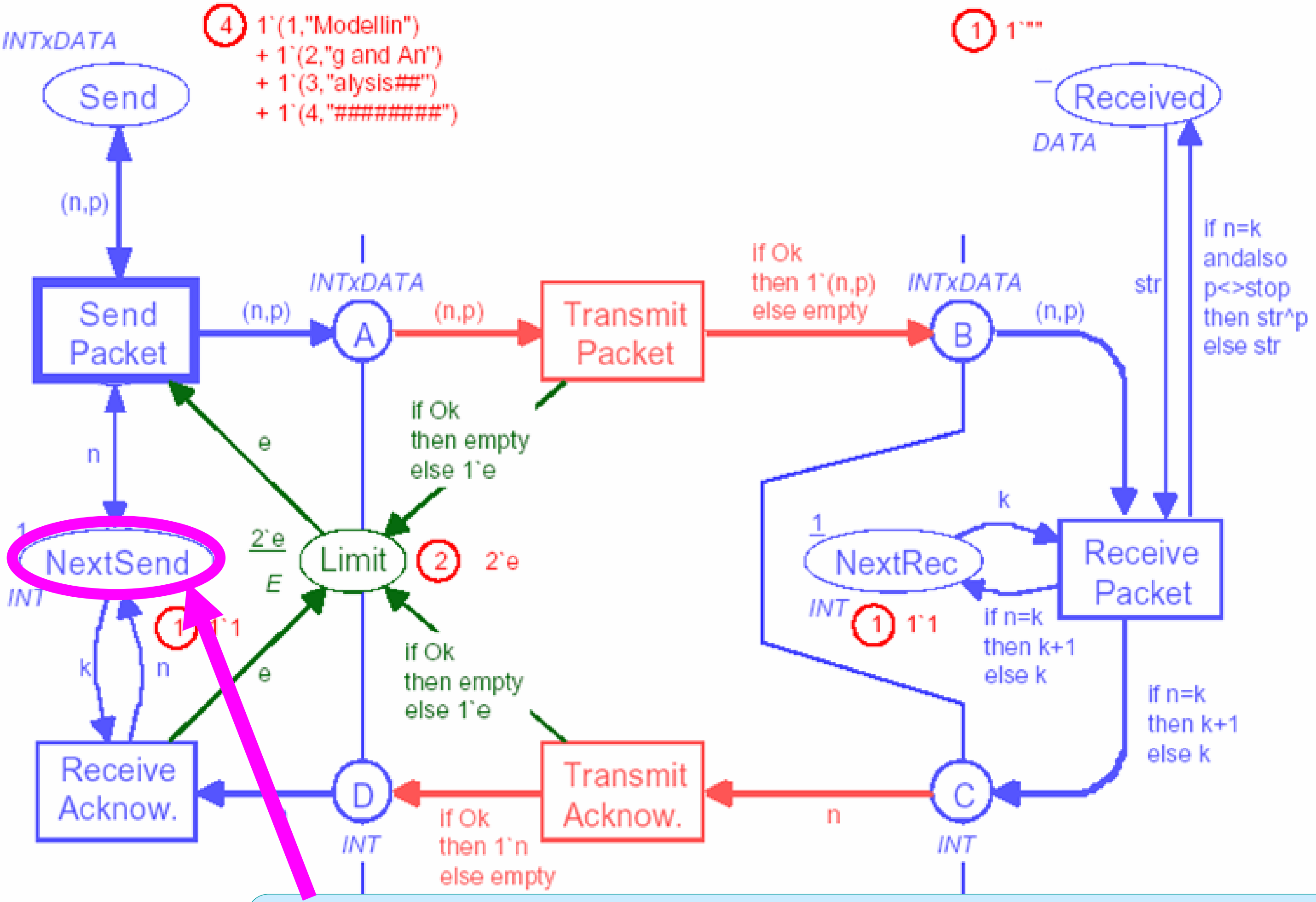
NextSend: 1`2
NextRec: 1`2
A: 1`(2,"g and An")
Received: 1`"Modellin"

10
1:2

NextSend: 1`1
NextRec: 1`2
B: 1`(1,"Modellin")
Received: 1`"Modellin"

9
1:1

Non-standard queries



Can the NextSend counter be decreased?

Query in Standard ML

```
PredAllArcs
      (fn a => ((ms_to_col(Mark.NextSend 1
                       (SourceNode a)))   >
              (ms_to_col(Mark.NextSend 1
                       (DestNode a))));
```

```
>[973,951,934,921,920,895,894,845,844,818,817,
 753,729,663,648,587,573,567,517,499,497,429,
 428,360,310,271,233] : Arc list
```

Yes!

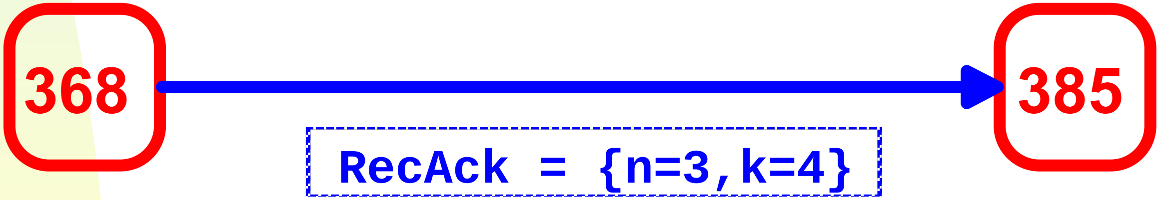
Counter example

```
DisplayArcs [973];
```

```
> () : unit
```

```
NextSend = 4
NextRec = 5
Received = "Modelling
and Analysis##"
B = 1`(4,"########")
D = 1`3
```

```
NextSend = 3
NextRec = 5
Received = "Modelling
and Analysis##"
B = 1`(4,"########")
```

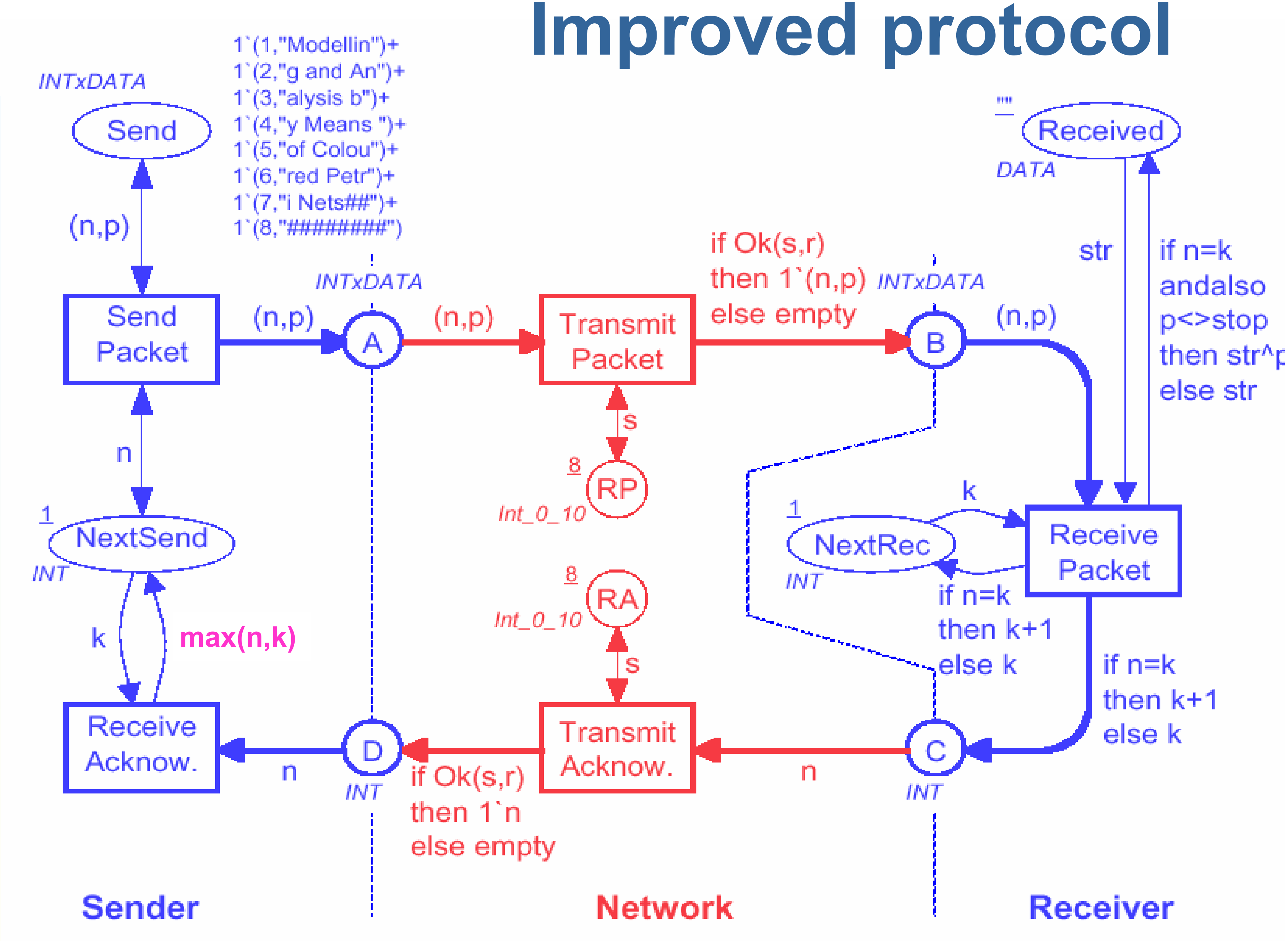


368

385

```
RecAck = {n=3,k=4}
```

Improved protocol



Temporal logic

- It is also possible to make *state space queries* by means of a CTL-like *temporal logic*.
  - *States.*
  - *Transitions.*
  - *Binding elements.*

State spaces - pro/contra

- 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.
  - The present version of our tool handles graphs with *one million* states.
  - For many systems this is *not sufficient*.

Statistics – full state spaces

Limit:		**1**	**2**	**3**	**4**	**5**	**6**
Nodes	Original	33	428	3,329	18,520	82,260	310,550
	Max	33	293	1,829	9,025	37,477	136,107
	Ratio	1.0	1.46	1.82	2.05	2.19	2.28
Arcs	Original	44	1,130	12,825	91,220	483,562	2,091,223
	Max	44	764	6,860	43,124	213,902	891,830
	Ratio	1.0	1.48	1.87	2.12	2.26	2.34
Secs	Original	0	0	3	41	560	7,686
	Max	0	0	2	16	153	1,634
	Ratio	-----	-----	1.5	2.56	3.66	4.70

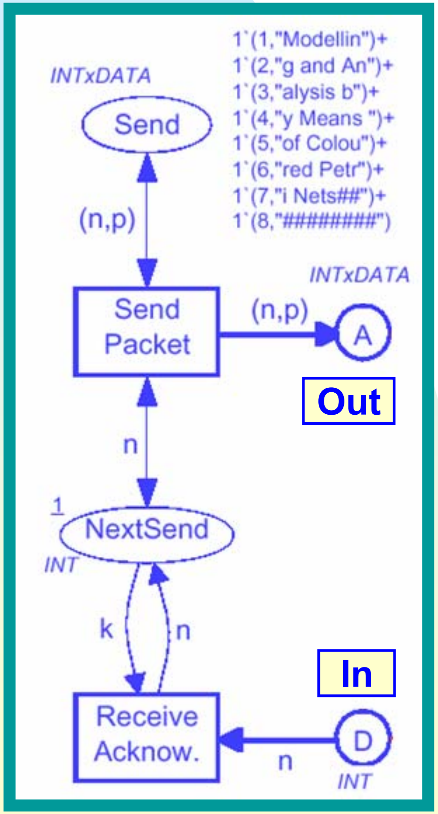
- Intel Pentium III, 1GHz, 1 GB RAM

Condensed state spaces

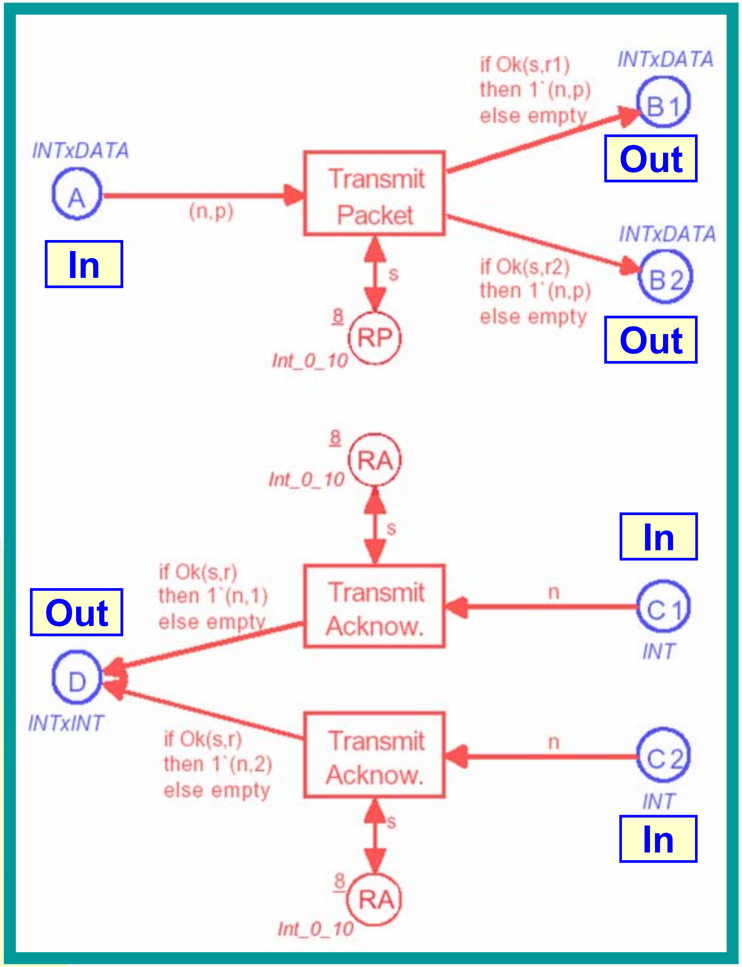
- Fortunately,it is sometimes possible to construct much more *compact* state spaces – *without loosing information*.
- This is done by exploiting:
  - *Symmetries* in the modelled system.
  - Other kinds of *equivalent behaviour.*
  - *Progress measure*.
  - *Concurrency* between events.

Protocol with multiple receivers

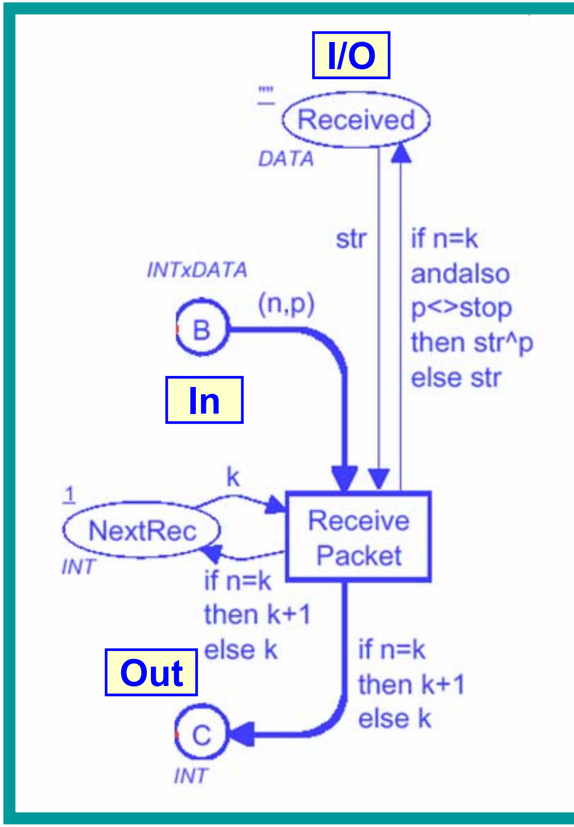
Sender



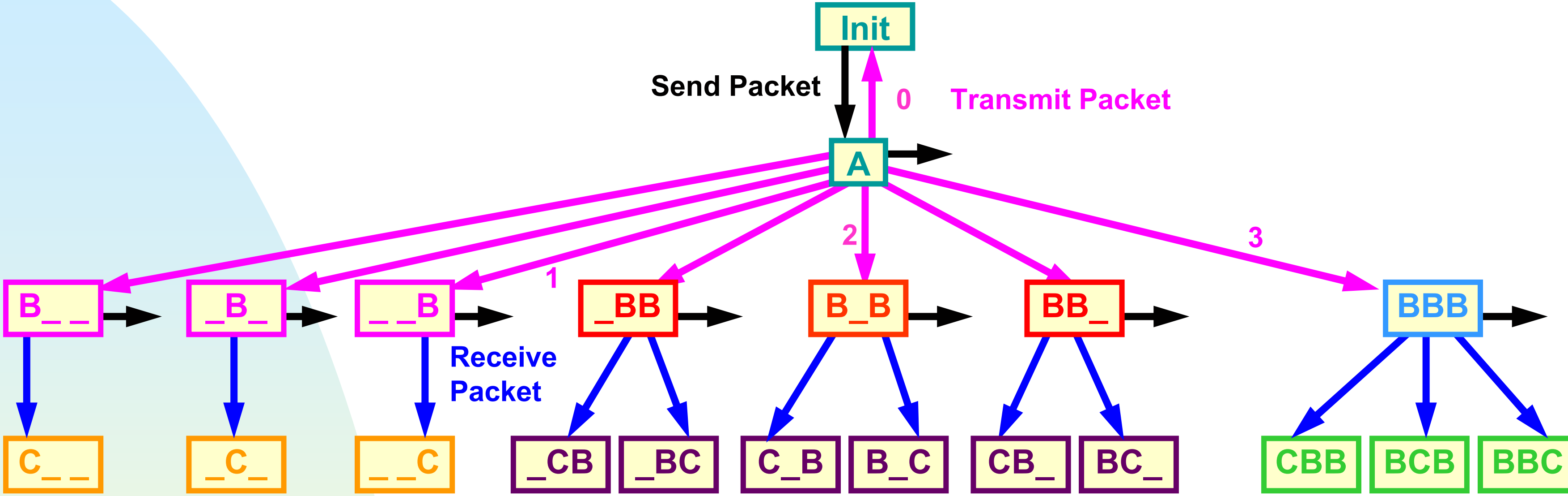
Network



Receiver

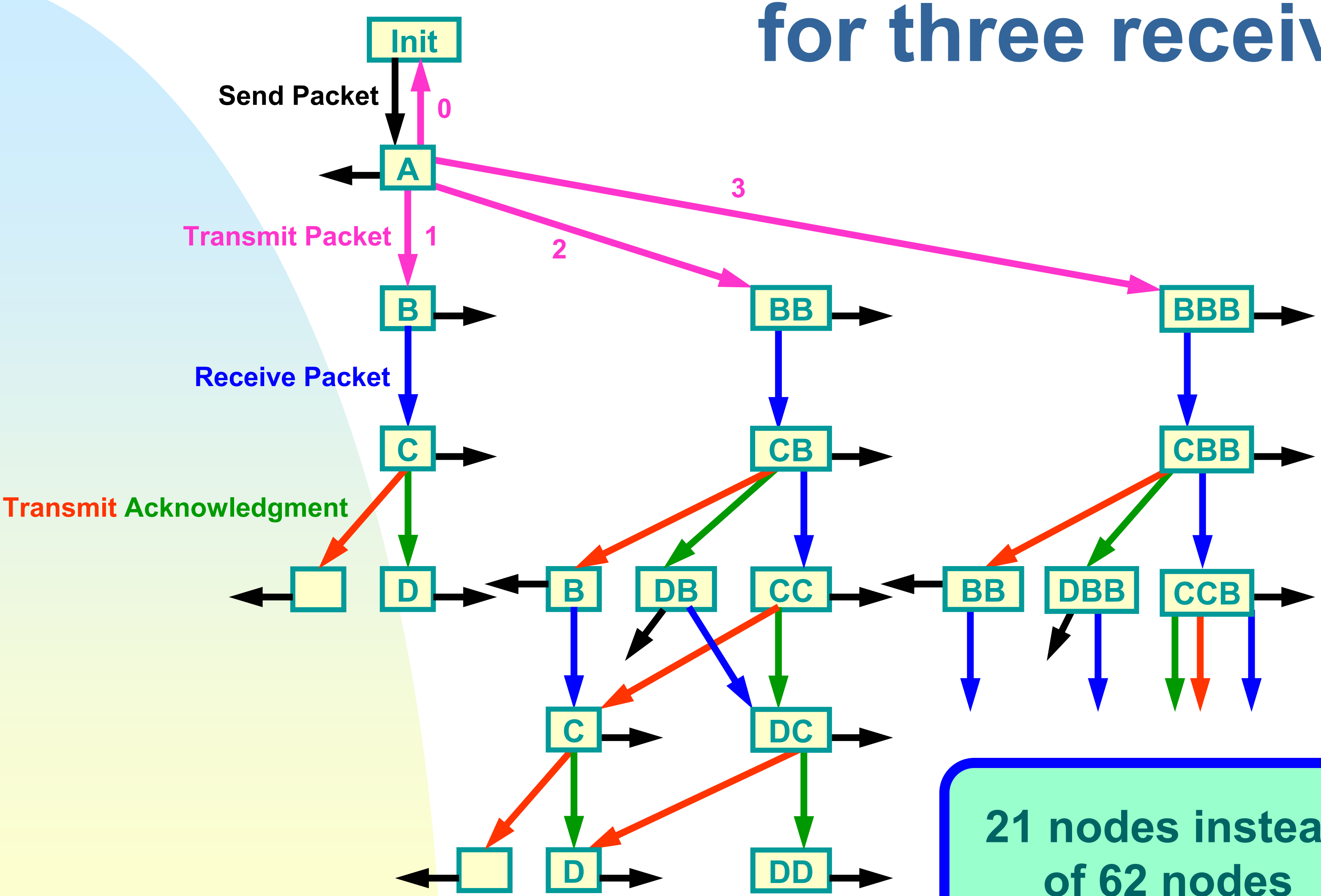


State space for three receivers



- The *red* nodes are *equivalent* (or *symmetrical*).
- They also have *equivalent:*
  - *direct successors,*
  - *enabled binding elements.*

Condensed state space for three receivers



21 nodes instead of 62 nodes

Symmetries

- A *symmetry* is a *function* ϕ that maps:
  - *markings* into equivalent *markings,*
  - *binding elements* into equivalent *binding elements.*

- A *symmetry specification* is a *set of functions* $\Phi \subseteq [\mathbf{M} \cup \mathbf{BE} \rightarrow \mathbf{M} \cup \mathbf{BE}]$ such that:
  - $\forall \phi \in \Phi: (\phi \mid \mathbf{M}) \in [\mathbf{M} \rightarrow \mathbf{M}] \wedge (\phi \mid \mathbf{BE}) \in [\mathbf{BE} \rightarrow \mathbf{BE}]$.
  - $(\Phi, \circ)$ is an *algebraic group.*

  Each element of Φ is called a *symmetry.*

Equivalent markings

- Two *markings* M and M* are *equivalent* iff there exist a *symmetry* ϕ that maps M* into M:

  $$M \approx_{M} M^* \quad \Leftrightarrow \quad \exists \phi \in \Phi : M = \phi(M^*).$$

- Two *binding elements* b and b* are *equivalent* iff there exist a *symmetry* ϕ that maps b* into b:

  $$M \approx_{BE} M^* \quad \Leftrightarrow \quad \exists \phi \in \Phi : b = \phi(b^*).$$

- $(\Phi, \circ)$ is an *algebraic group.* This *implies* that $\approx_{M}$ and $\approx_{BE}$ are *equivalence relations.*

Consistency

- We demand that *equivalent markings* must have:
  - *equivalent direct successors,*
  - *equivalent enabled binding elements.*

- A *symmetry specification* Φ is *consistent* iff the following properties are satisfied for all symmetries $\phi \in \Phi$, all reachable markings M_1, M_2 and all binding elements b:
  - $M_1 \xrightarrow{b} M_2 \quad \Leftrightarrow \quad \phi(M_1) \xrightarrow{\phi(b)} \phi(M_2)$.
  - $\phi(M_0) = M_0$.

Protocol with multiple receivers

- *Symmetries* are defined as *consistent permutations* of *receiver-IDs:*
  - When we model each receiver by a *separate module* we permute the *markings* of these modules.
  - When we model all receivers by a single module (adding a *new component* to the token colours) we permute the *colour values* in the type:

    REC = {rec_1, rec_2, rec_3,…}.

Construction of state spaces with symmetries

- State spaces with *symmetries* are *constructed* in the same way as *ordinary state spaces,* except that:
  - Before *adding a new node* we check whether the marking is *equivalent* to the marking of an existing node.
  - Before *adding a new arc* we check whether the binding element is *equivalent* to the binding element of an existing arc (from the same source node).

What can we prove from state spaces with symmetries?

- State spaces with *symmetries* can be used to *investigate* the same kinds of *behavioural properties* as ordinary state spaces, but only *modulo equivalence.*
- As an example, this means that:
  - We *cannot* investigate whether a certain *marking* is reachable *itself.*
  - Instead we *can* investigate whether there is an *equivalent marking* which is reachable.

Statistics – symmetries

Limit = Receivers		**2**	**3**	**4** **(3 packets)**	**5** **(2 packets)**	**6** **(2 packets)**
Nodes	Full	921	22,371	172,581	486,767	5,917,145
	Sym	477	4,195	9,888	8,387	24,122
	Ratio	1.93	5.33	17.45	58.04	245.30
Arcs	Full	1,832	64,684	671,948	2,392,458	35,068,448
	Sym	924	11,280	32,963	31,110	101,240
	Ratio	1.98	5.73	20.38	76.90	346.39
Time	Full	2 secs	4 mins	191 mins	-----	-----
	Sym	3 secs	2 mins	8 mins	8 mins	1 hour
	Ratio	0.7	2.0	23.9	-----	-----
Perms	n!	2	6	24	120	720

- Prototype implementation in 1998.

We can be more general

- We have defined the *equivalence relations* for markings and bindings elements from a *set of symmetry functions.*
- Instead we may define the *equivalence relations directly* (i.e. from scratch).
- An *equivalence specification* is a pair $(\approx_M, \approx_{BE})$ where:
  - $\approx_M$ is an *equivalence relation* on the set of *all markings.*
  - $\approx_{BE}$ is an *equivalence relation* on the set of *all binding elements.*

Consistency

- As before, we demand that *equivalent markings* must have:
  - *equivalent direct successors,*
  - *equivalent enabled binding elements.*

- An *equivalence specification* $(\approx_M, \approx_{BE})$ is *consistent* iff for all reachable markings M_1, M_2, M and all binding elements b:

$$M_1 \approx_M M_2 \;\wedge\; M_1 \xrightarrow{b} M \;\Rightarrow$$

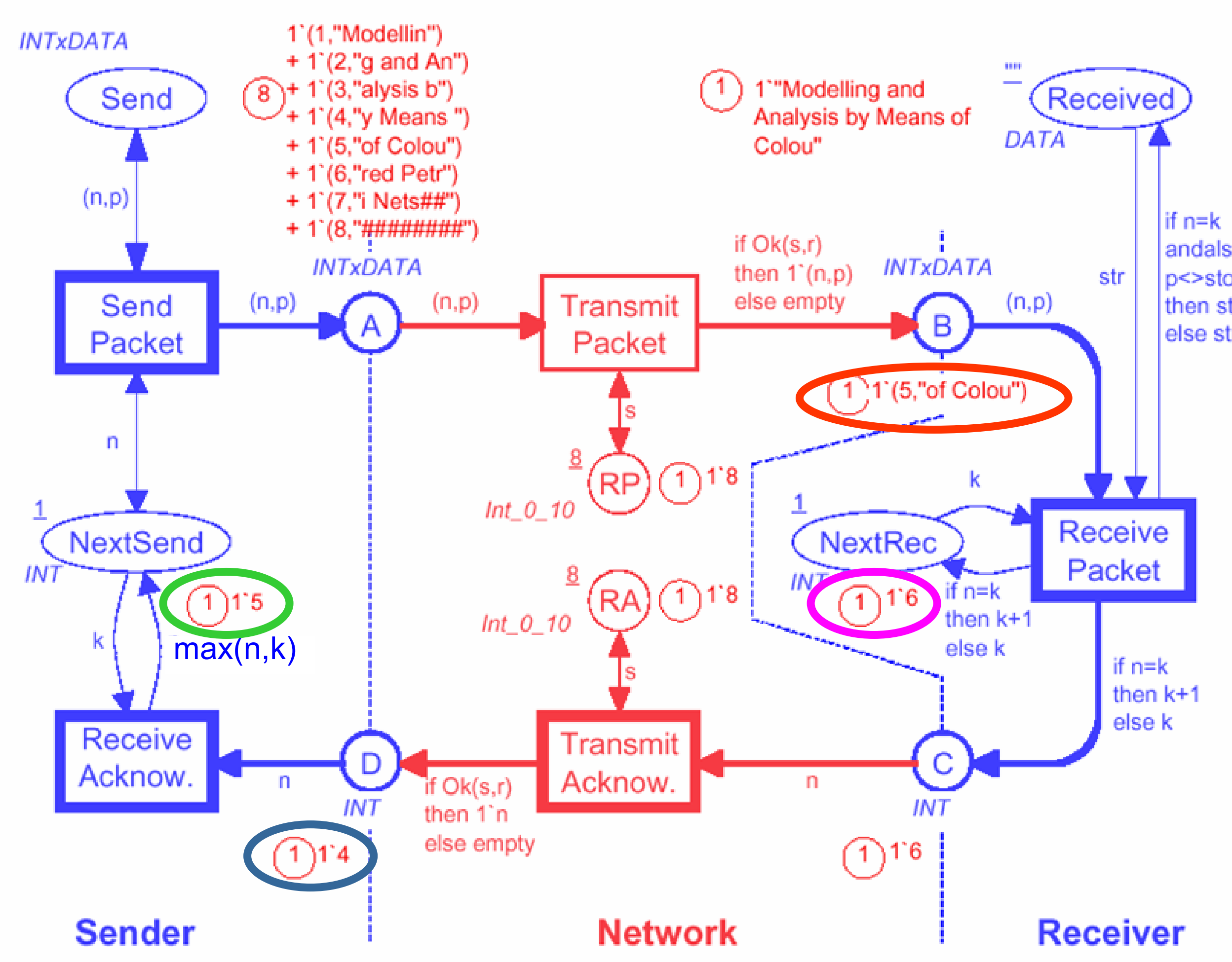
$$\exists M^* \approx_M M \; \exists b^* \approx_{BE} b: M_2 \xrightarrow{b^*} M^*.$$

State spaces with equivalence classes

- State spaces with *equivalence classes* are *constructed* in the same way as state spaces with *symmetries*.
- They can be used to *investigate* the same kinds of *behavioural properties.*
- State spaces with *symmetries* is a *special case* of state spaces with *equivalence classes.*

Intermediate state of protocol

- *Receiver* expects packet no. 6.
- *Sender* is still sending packet no. 5.
- This *packet* will be ignored.
  It is ***old.***
- This *acknowledgment* will also be ignored.
  It is ***old.***



Equivalence relation

- A marking $M(p)$ where p is one of the network places A,B,C,D is *split* into two parts:
  - $M(p) = M(p)_{OLD} + M(p)_{NEW}$

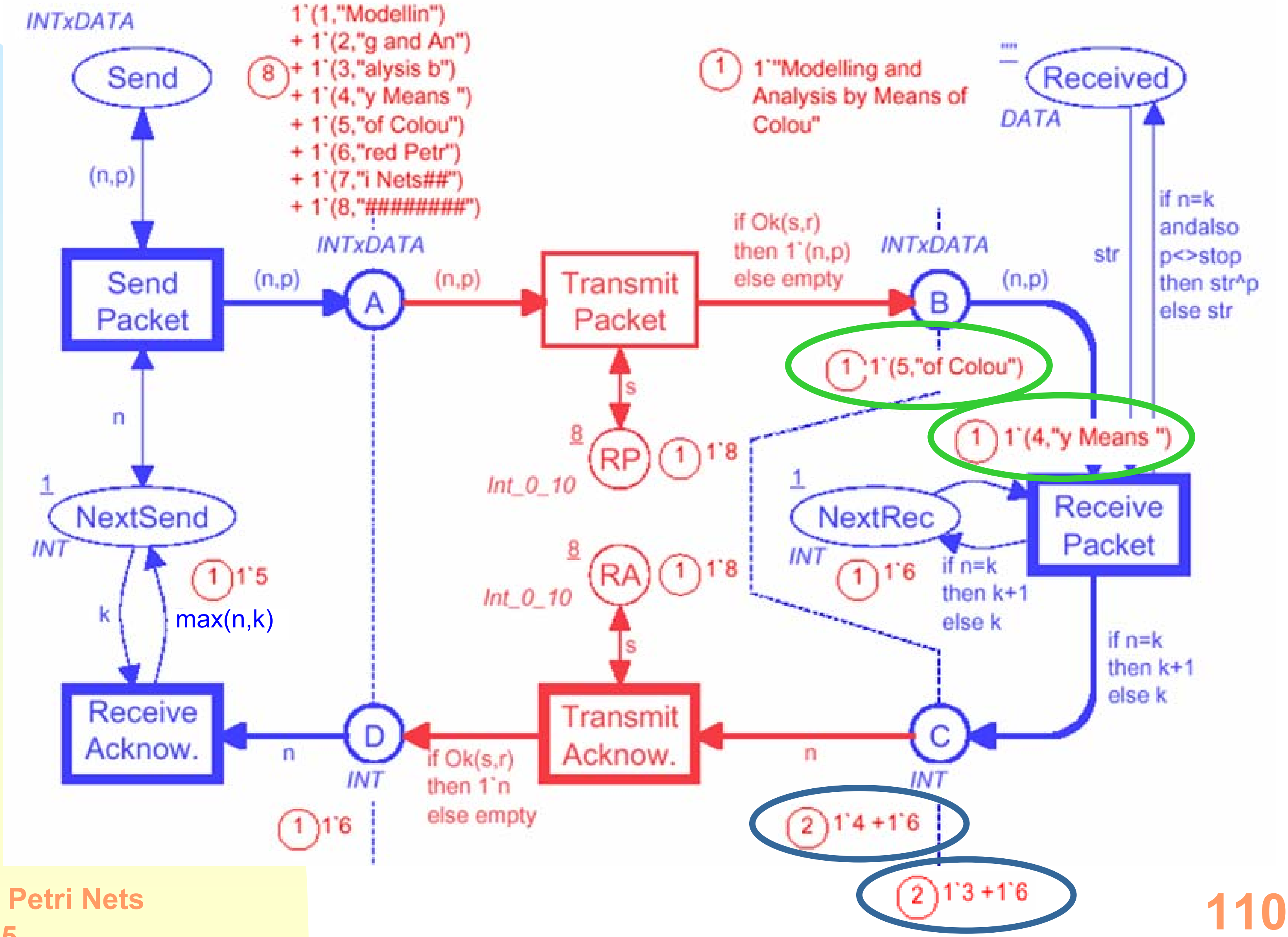
Old packets/acks

All remaining packets/acks

- Two markings M_1 and M_2 are *equivalent* iff:
  - $M_1(p) = M_2(p)$ for $p \notin \{A,B,C,D\}$

$$\left.\begin{array}{l} |M_1(p)_{OLD}| = |M_2(p)_{OLD}| \\ M_1(p)_{NEW} = M_2(p)_{NEW} \end{array}\right\} \text{for } p \in \{A,B,C,D\}$$

Two equivalent states



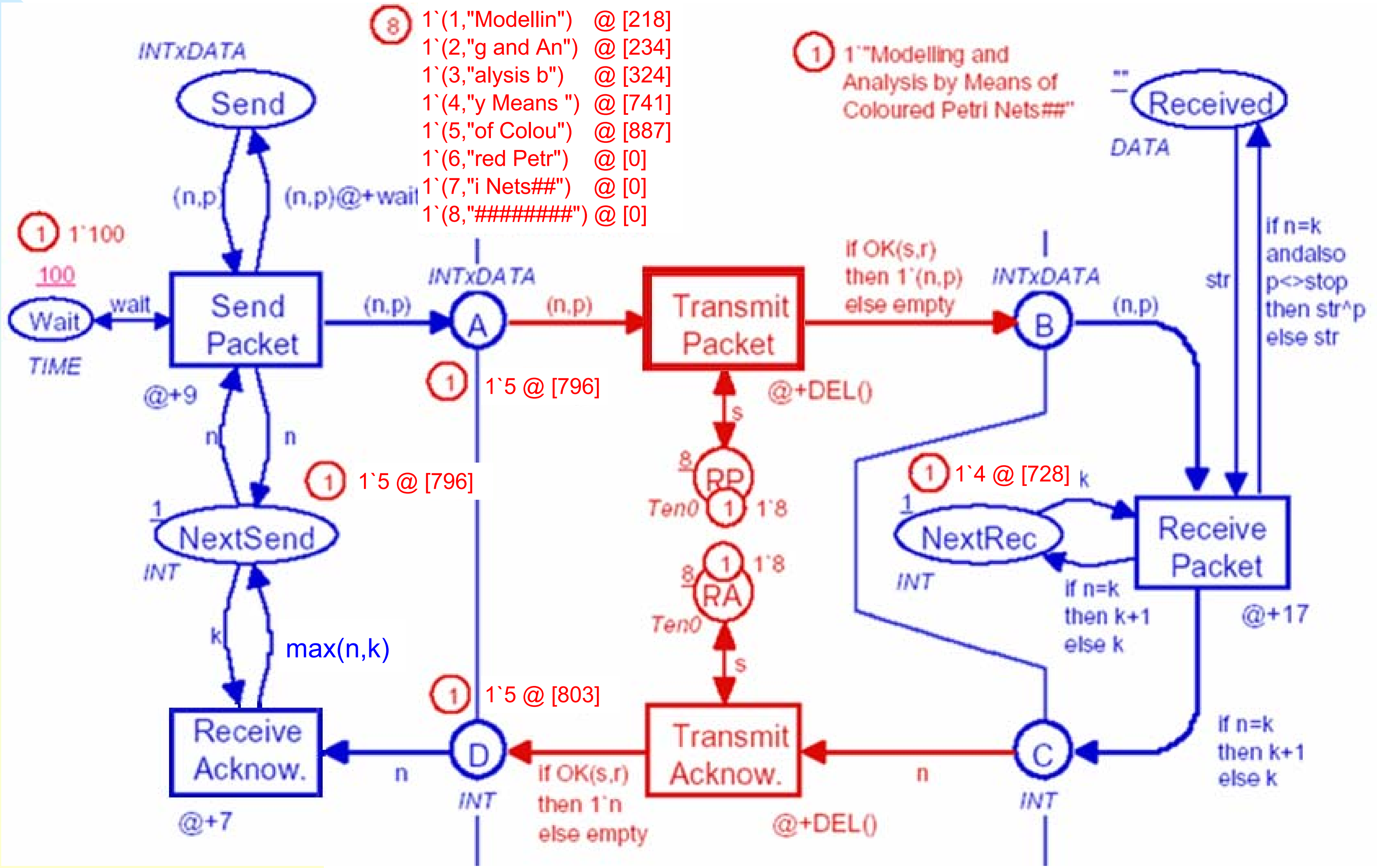
Statistics – equivalence classes

Limit:		**1**	**2**	**3**	**4**	**5**	**6**
Nodes	Full	33	293	1,829	9,025	37,477	136,107
	Equiv	33	155	492	1,260	2,803	5,635
	Ratio	1.0	1.89	3.72	7.16	13.37	24.15
Arcs	Full	44	764	6,860	43,124	213,902	891,830
	Equiv	44	383	1,632	5,019	12,685	28,044
	Ratio	1.0	1.99	4.20	8.59	16.86	31.80
Secs	Full	1	1	6	56	642	7,507
	Equiv	1	1	7	36	157	553
	Ratio	1.0	1.0	0.9	1.56	4.09	13.58

- Sun Ultra Sparc 3000, 512 MB in 1997.

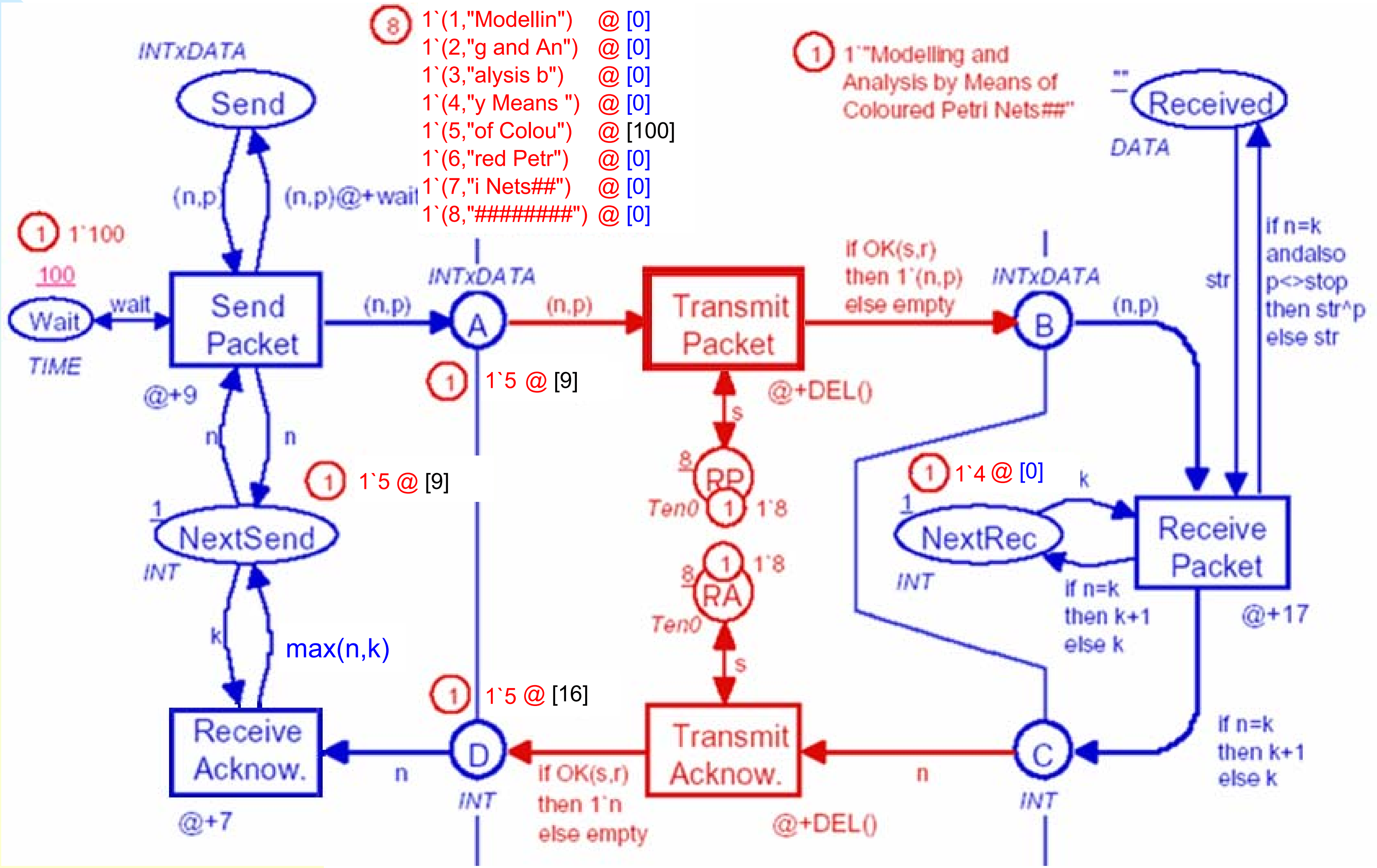
Timed protocol

Creation time 787 → 0

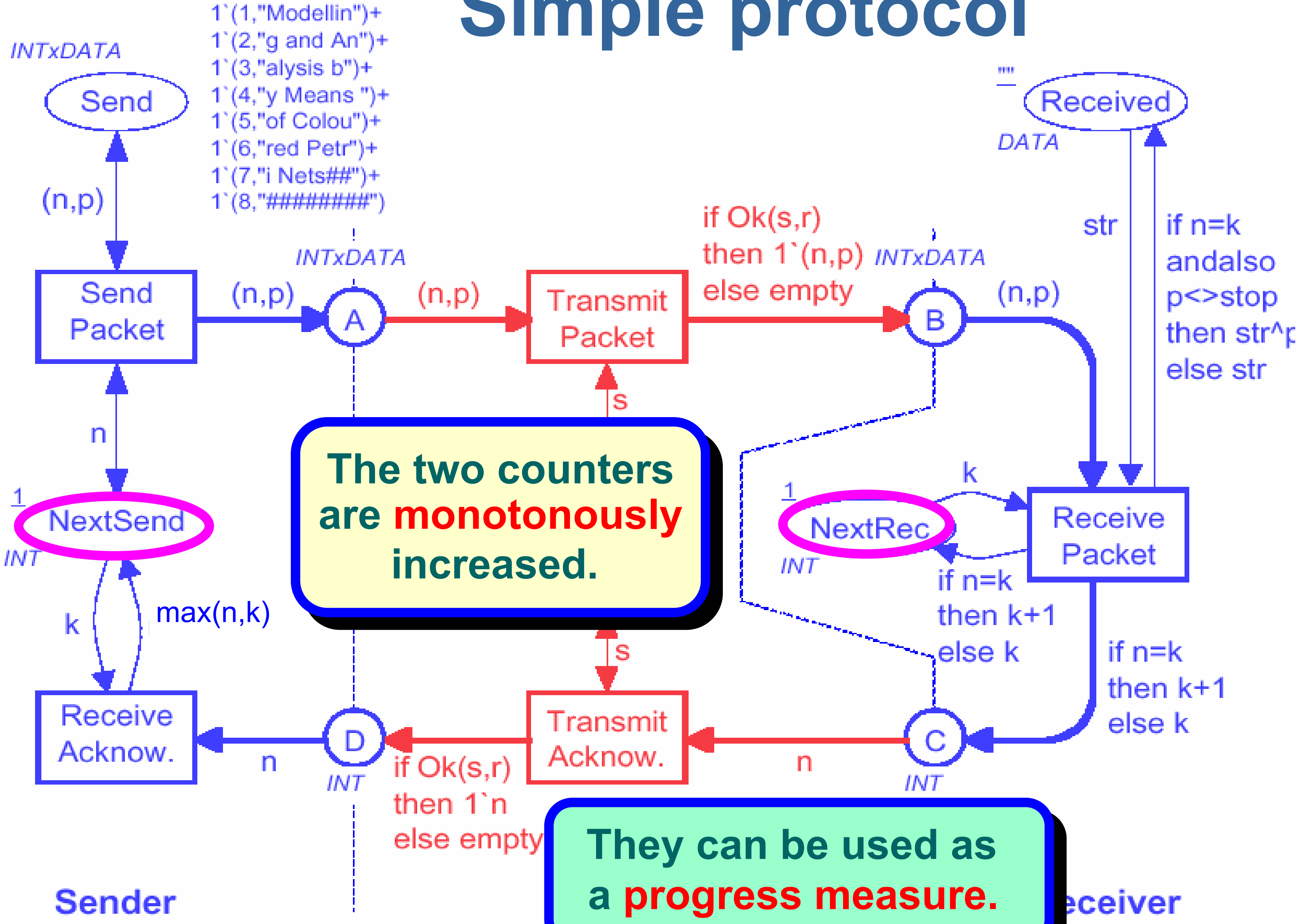


Timed protocol

Creation time 787 → 0



Simple protocol



The two counters are monotonously increased.

They can be used as a progress measure.

Progress measure

- **PM : STATES → A**

function

all states

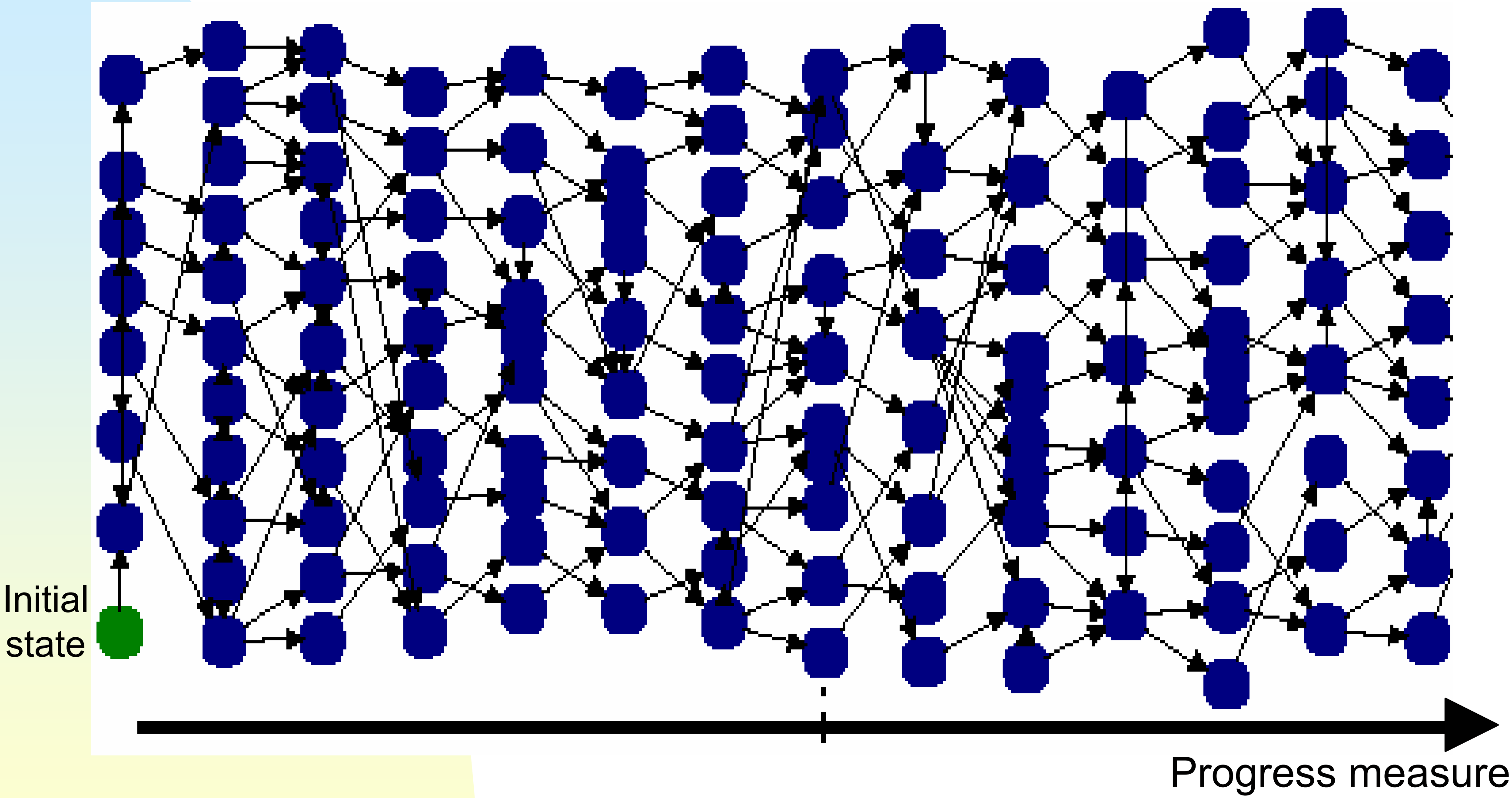
set with linear
or partial order ≤

- *Monotonous (non-decreasing):*

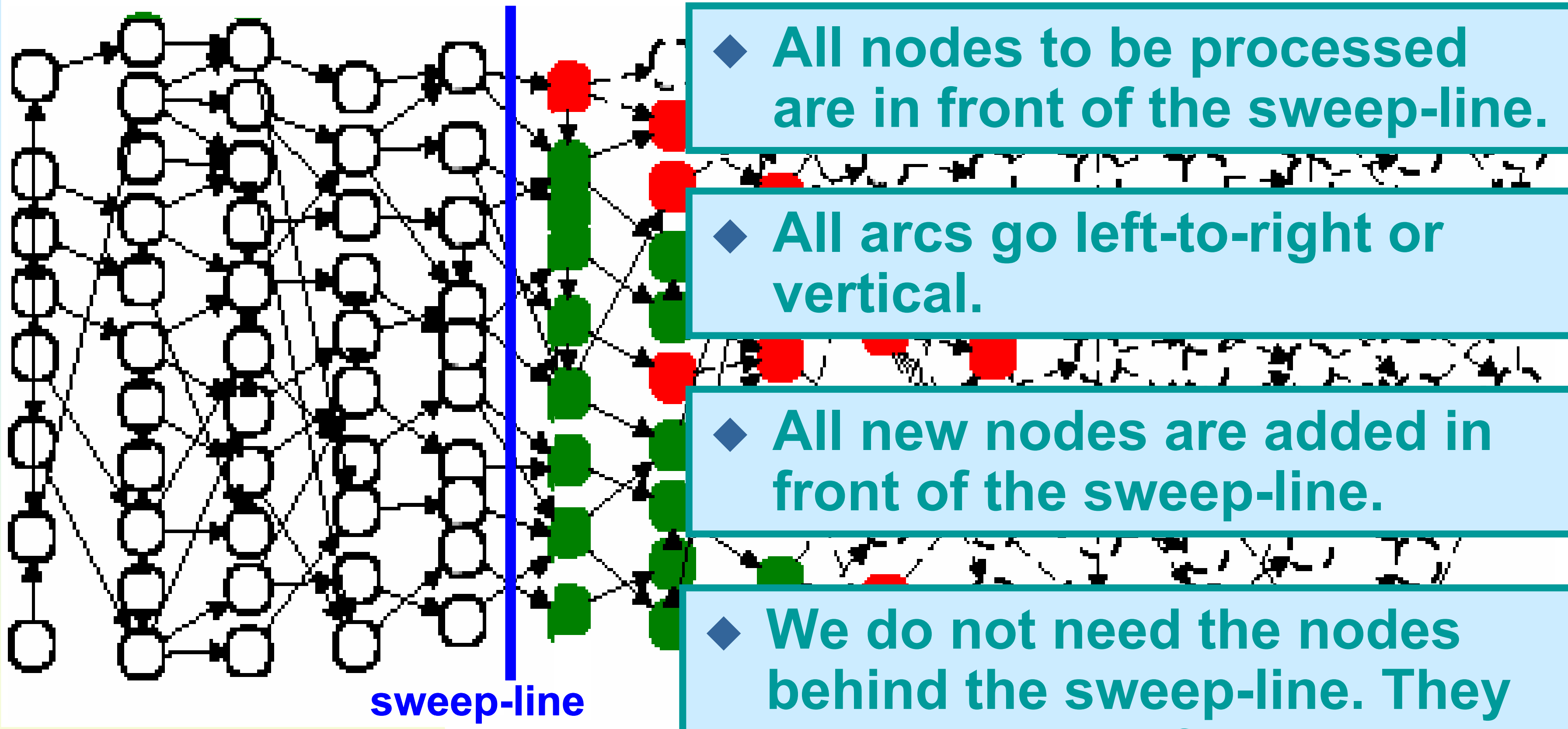
PM(X) ≤ PM(Y)

- Protocol: (NextSend,NextRec)
  lexicographical ordering.

States sorted by progress measure



Construction of state space

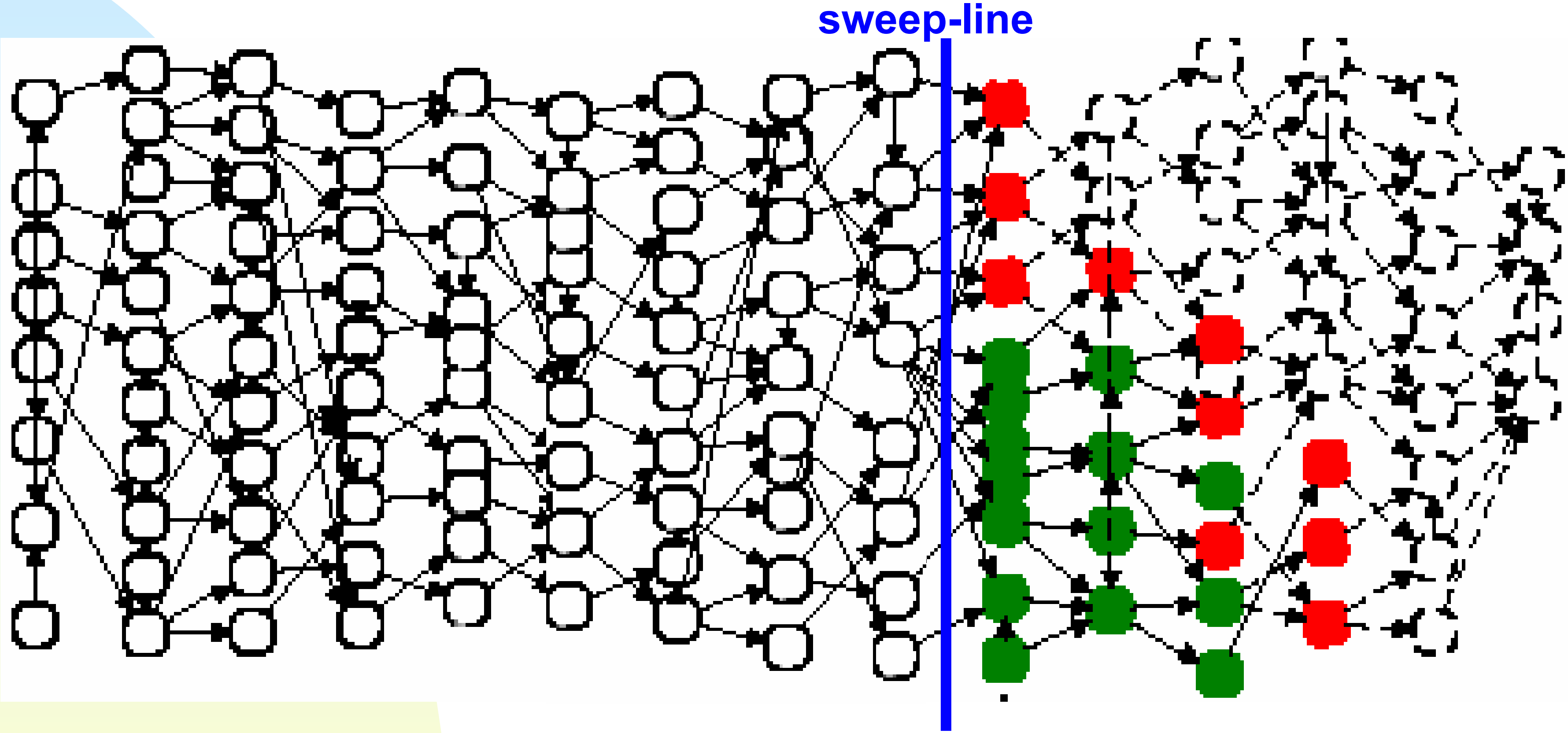


- ◆ All nodes to be processed are in front of the sweep-line.
- ◆ All arcs go left-to-right or vertical.
- ◆ All new nodes are added in front of the sweep-line.
- ◆ We do not need the nodes behind the sweep-line. They can be deleted from memory.

🟩 Processed

🟥 Unprocessed

We continue the construction



- The *sweep-line* moves from left to right.
  - In front of it, we *add new nodes.*
  - Behind it, we *remove nodes.*

Statistics – sweep-line

Limit:		1	2	3	4	5	6
Nodes	Full	33	293	1,829	9,025	37,477	136,107
	Sweep	33	134	758	4,449	20,826	82,586
	Ratio	1.0	2.19	2.41	2.03	1.80	1.65
Arcs	Full	44	764	6,860	43,124	213,902	891,830
	Sweep	-----	-----	-----	-----	-----	-----
	Ratio	-----	-----	-----	-----	-----	-----
Secs	Full	0	0	2	16	153	1,634
	Sweep	0	0	0	9	93	1,083
	Ratio	-----	-----	-----	1.78	1.65	1.51

- Intel Pentium III, 1GHz, 1 GB RAM

Statistics – sweep-line

Limit = 4

Packets:		**4**	**5**	**6**	**7**	**8**
Nodes	Full	9,025	20,016	38,885	68,720	113,121
	Sweep	4,449	8,521	14,545	22,905	33,985
	Ratio	2.03	2.35	2.67	3.00	3.33
Arcs	Full	43,124	99,355	198,150	356,965	596,264
	Sweep	-----	-----	-----	-----	-----
	Ratio	-----	-----	-----	-----	-----
Secs	Full	12	41	125	345	864
	Sweep	7	21	57	152	359
	Ratio	1.71	1.95	2.19	2.27	2.41

- AMD Athlon 1.33GHz, 512 MB RAM

Sweep-line method – pro/contra

- We can construct *larger state spaces*, since we do not need to have all states in *memory at the same time*.
- In a *timed CP-net* we can use the *global clock* as a *progress measure* – *time does not go backwards*.
- *“Problems”:*
  - *Analysis* must be done *on the-fly.*
  - To deal with *reactive systems* we need to be able to use *non-monotonous* progress measures.
  - *Counter examples* are *more difficult* to construct, since part of the state space has been *deleted from memory.*

Overview of talk

Modelling

- Basic language
  - syntax
  - semantics
- Extensions
  - modules
  - time
- Tool support
  - editing
  - simulation

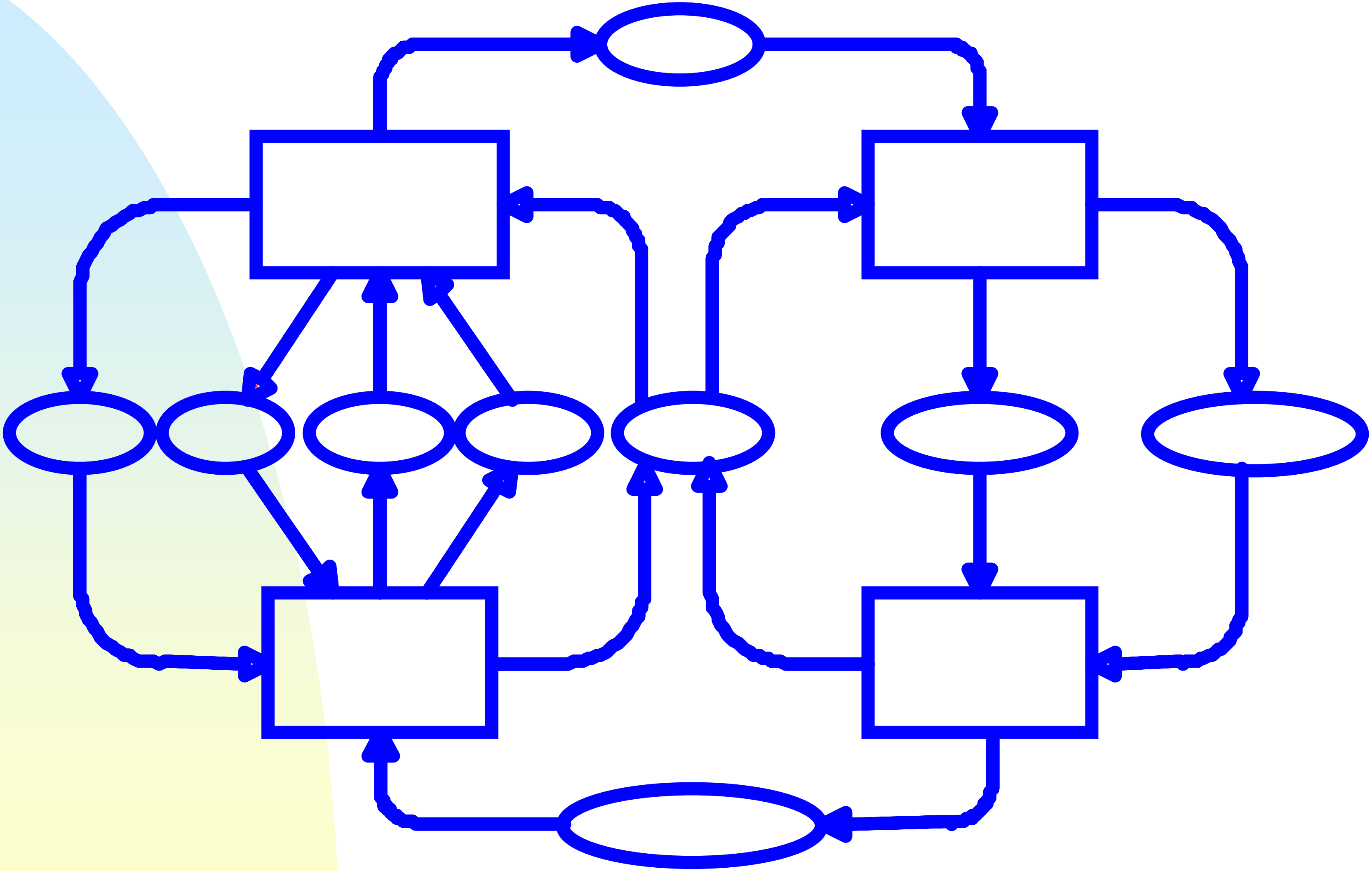
Analysis

- State spaces
  - full
  - symmetries
  - equivalence classes
  - sweep-line
- **Place invariants**
  - check of invariants
  - use of invariants

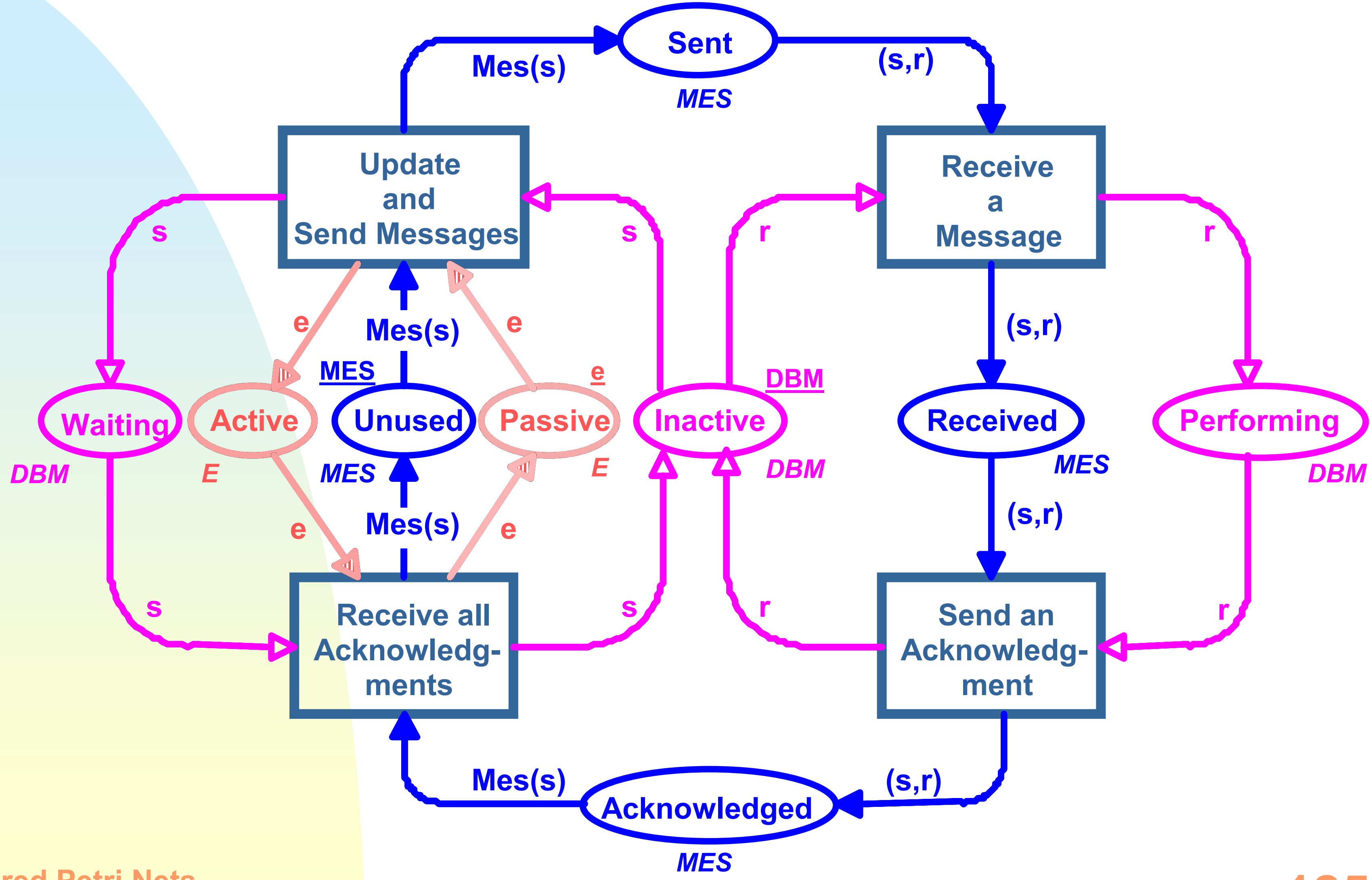
Place invariants

- The *basic idea* is similar to the use of invariants in *program verification.*
- An *invariant* describes a *property* which is *fulfilled for all reachable states.*
    - We first *construct* a set of place invariants.
    - Then we *check* whether they are fulfilled.
    - Finally, we *use* the place invariants to *prove behavioural properties* of the CP-net.

Logo of Petri net community

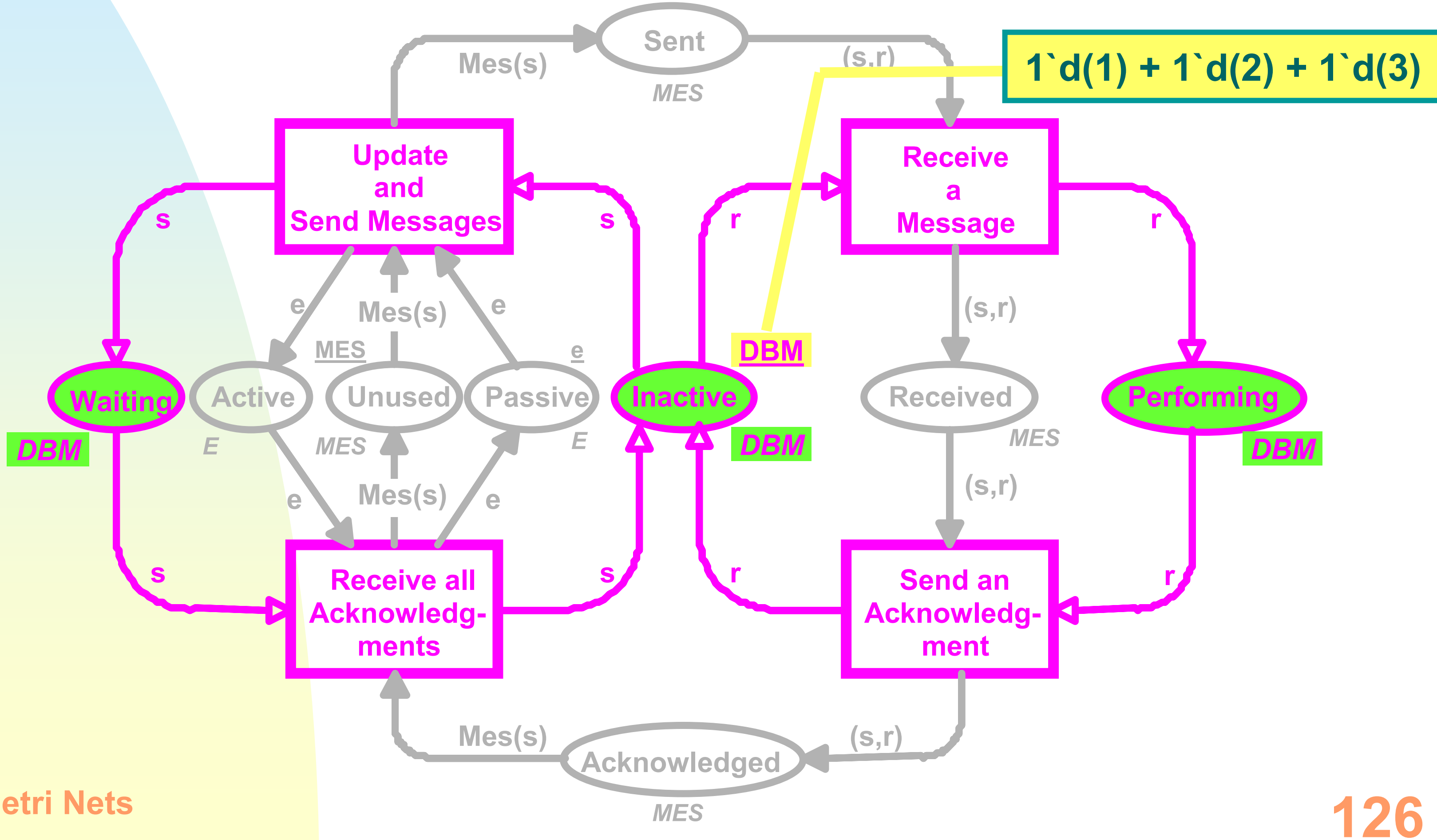


Distributed data base



Data base managers

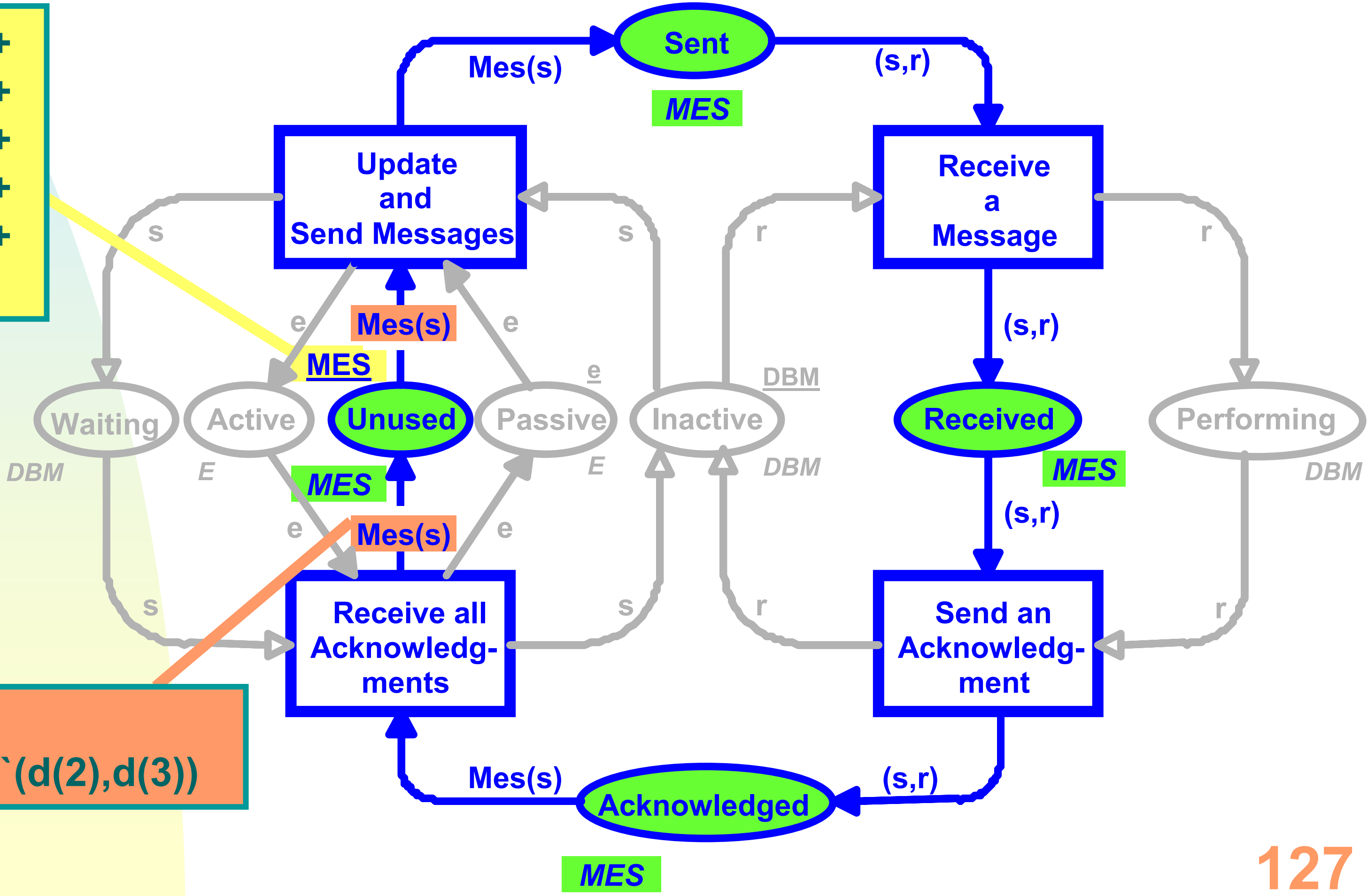
DBM = {d(1),d(2),d(3)}



Message buffers

MES = {(s,r)∈ DBM×DBM | s ≠ r}

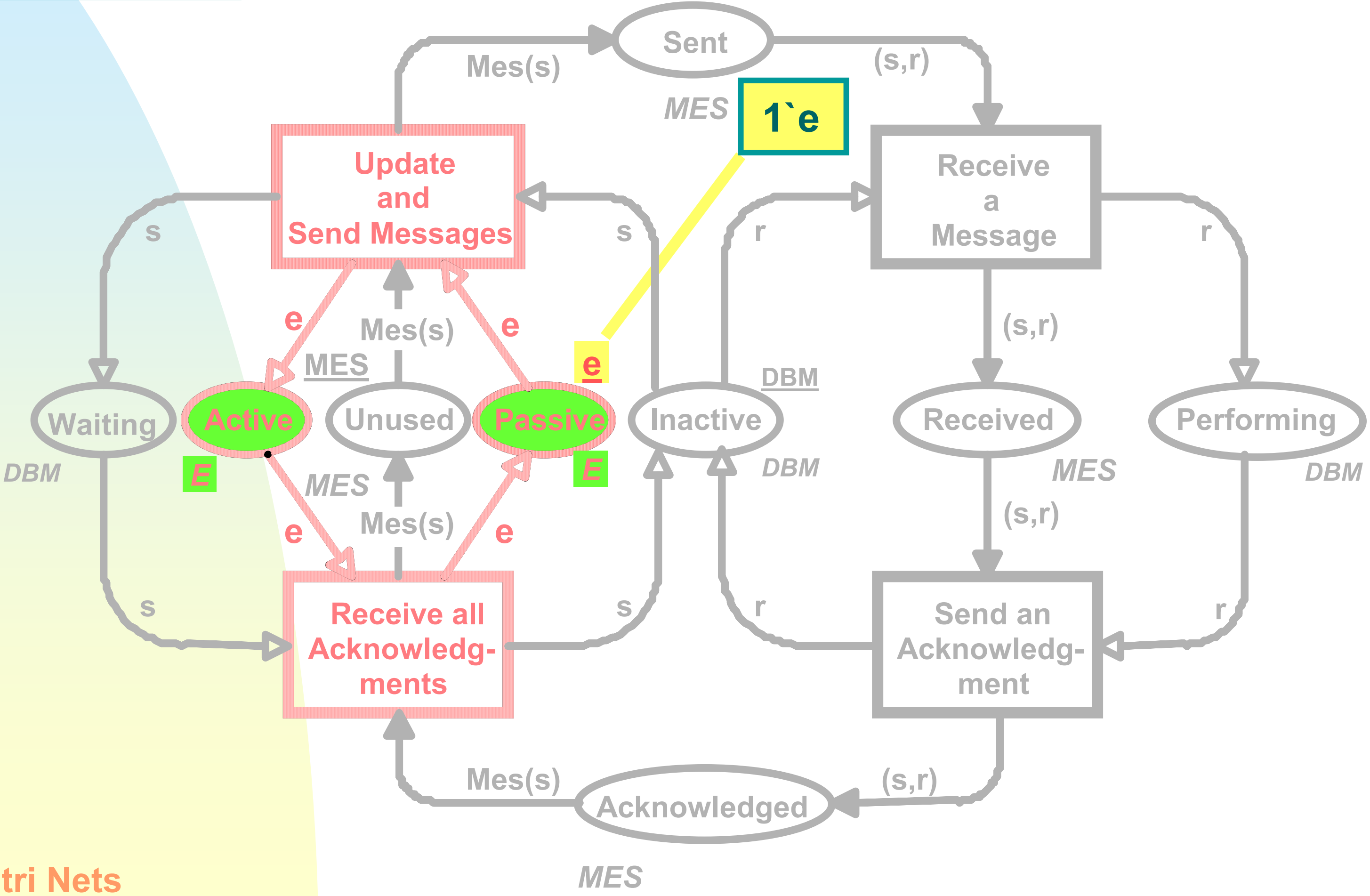
1\`(d(1),d(2)) +
1\`(d(1),d(3)) +
1\`(d(2),d(1)) +
1\`(d(2),d(3)) +
1\`(d(3),d(1)) +
1\`(d(3),d(2))



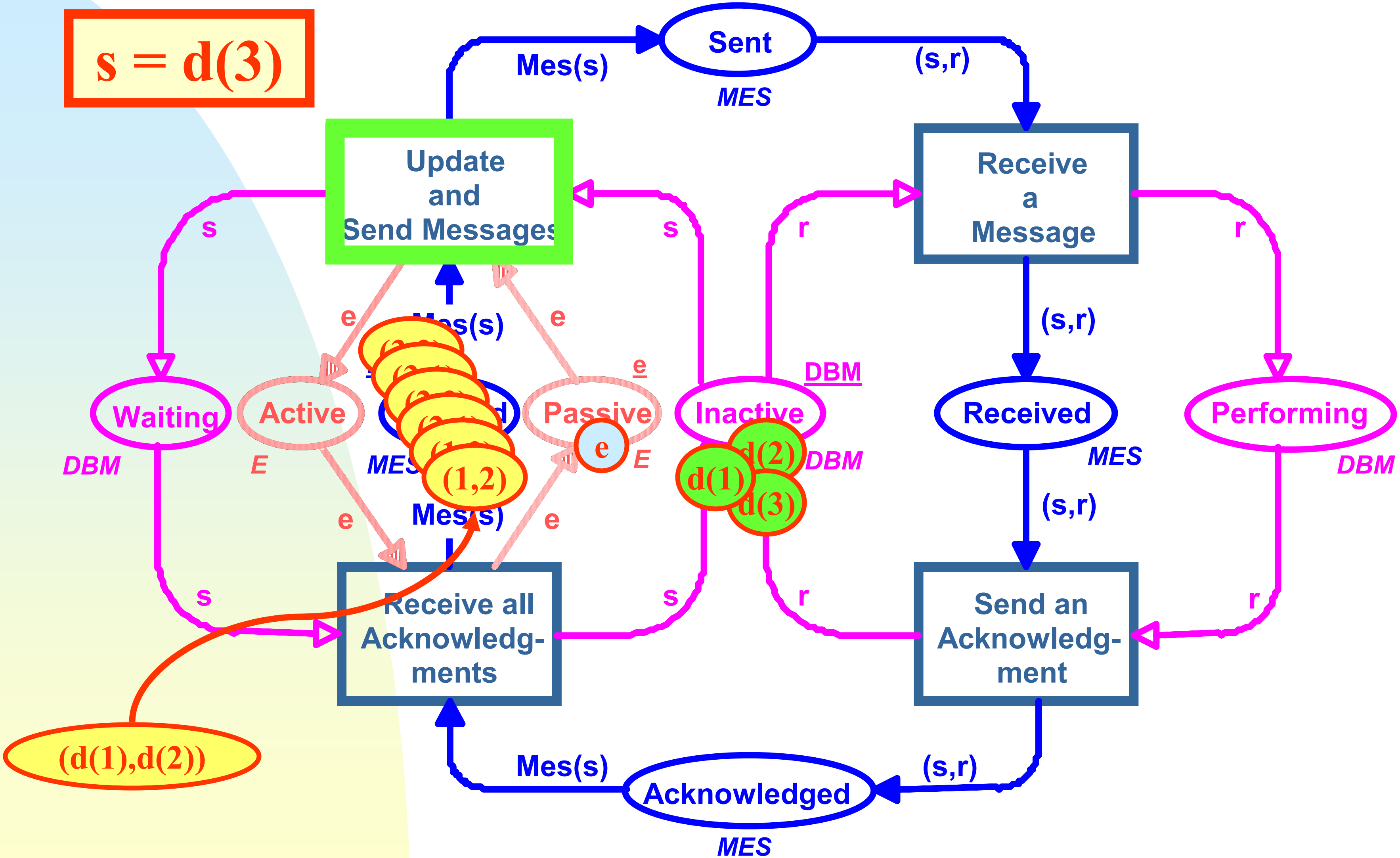
Mes(d(2)) =
1\`(d(2),d(1)) + 1\`(d(2),d(3))

Mutual exclusion

E = {e}

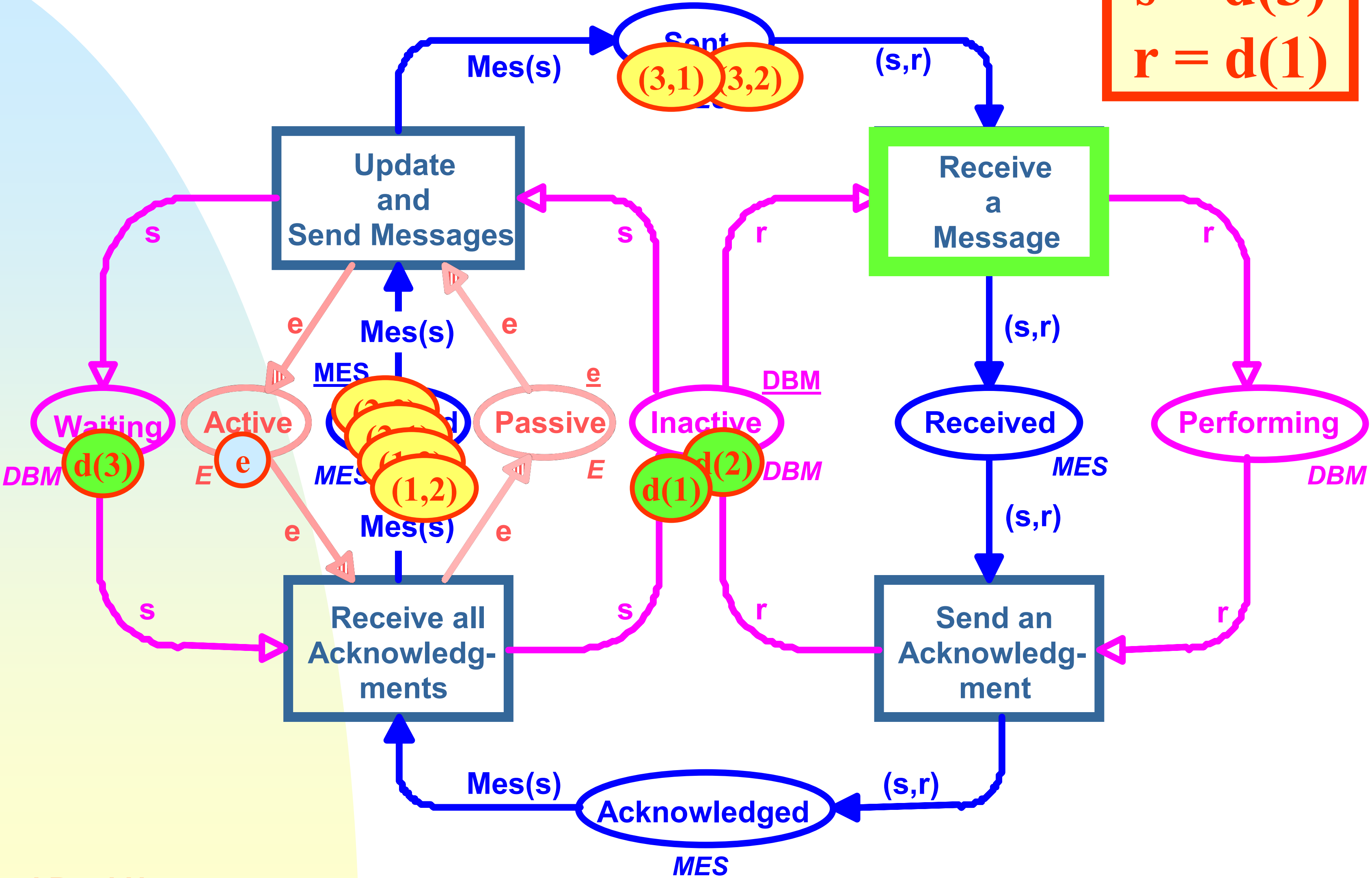


Distributed data base



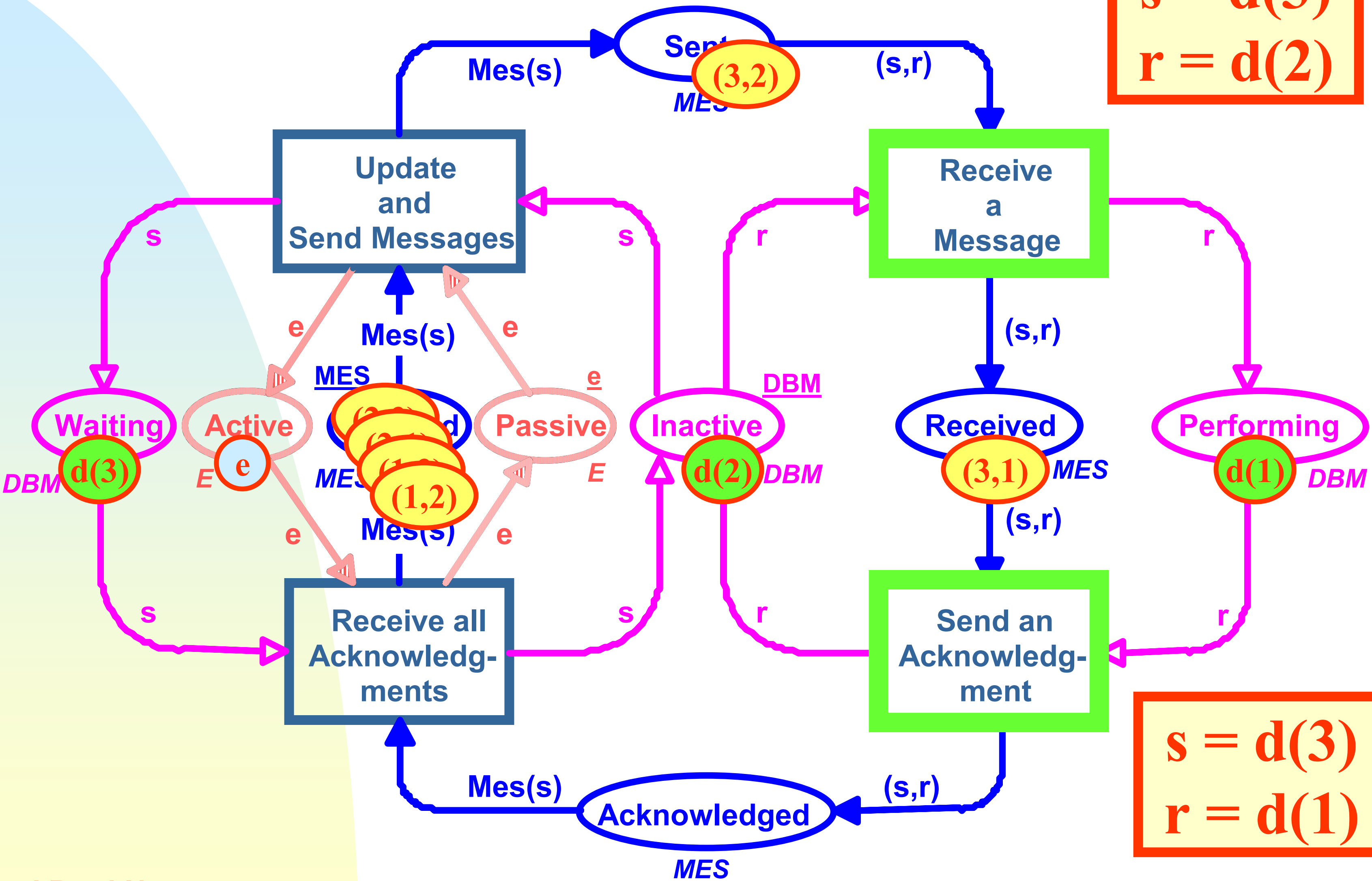
Distributed data base

s = d(3)
r = d(1)



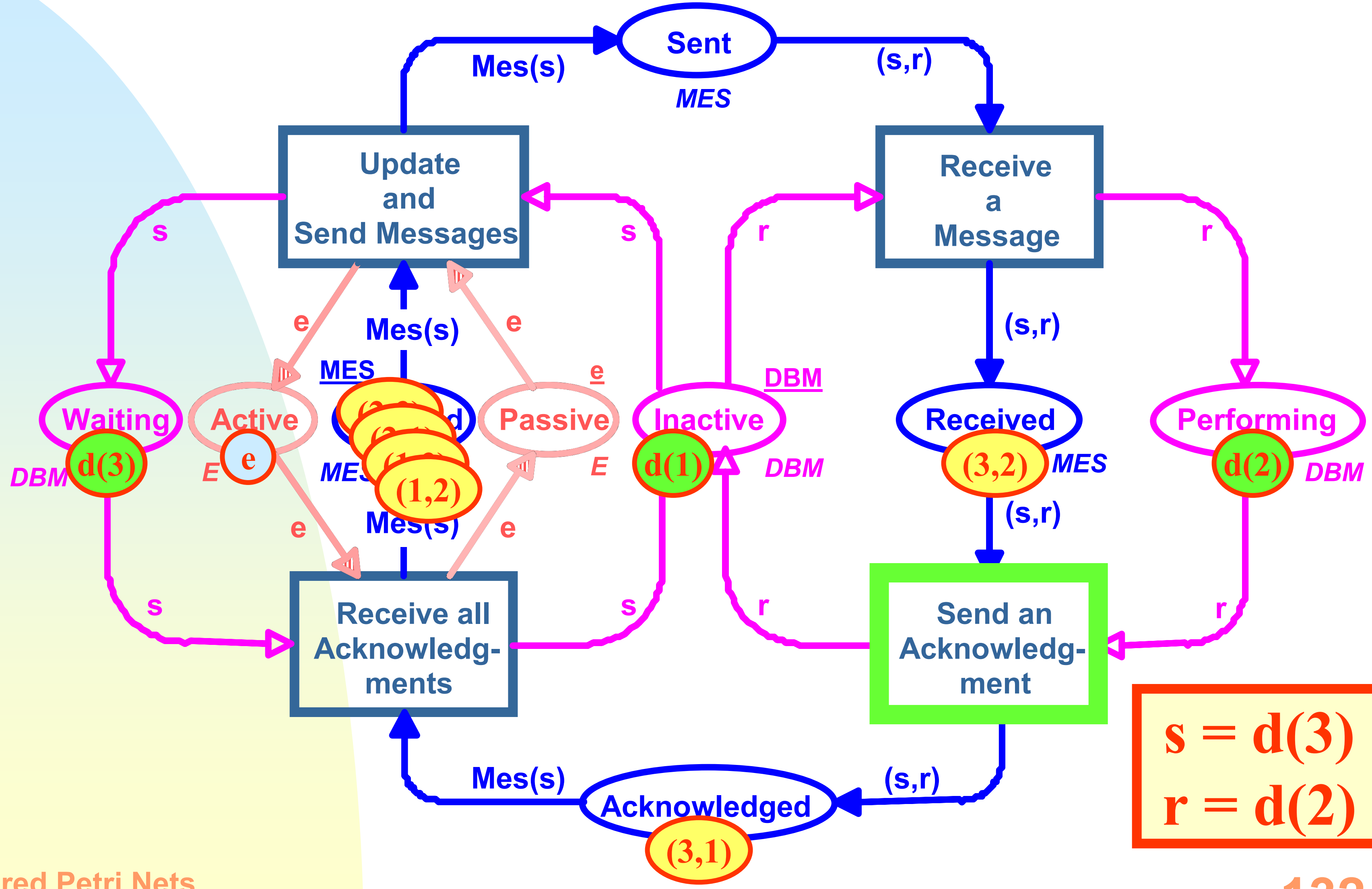
Distributed data base

s = d(3)
r = d(2)



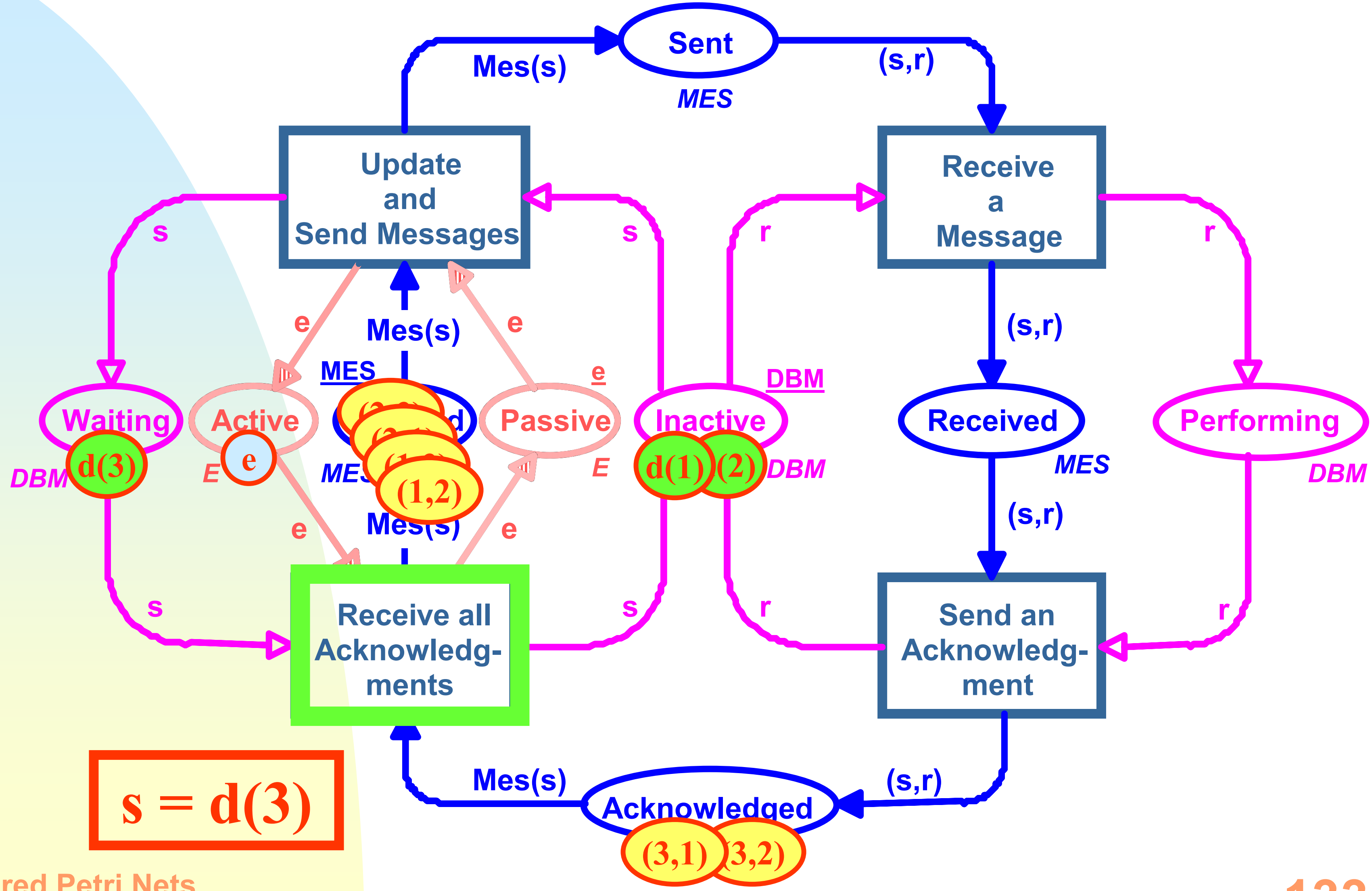
s = d(3)
r = d(1)

Distributed data base



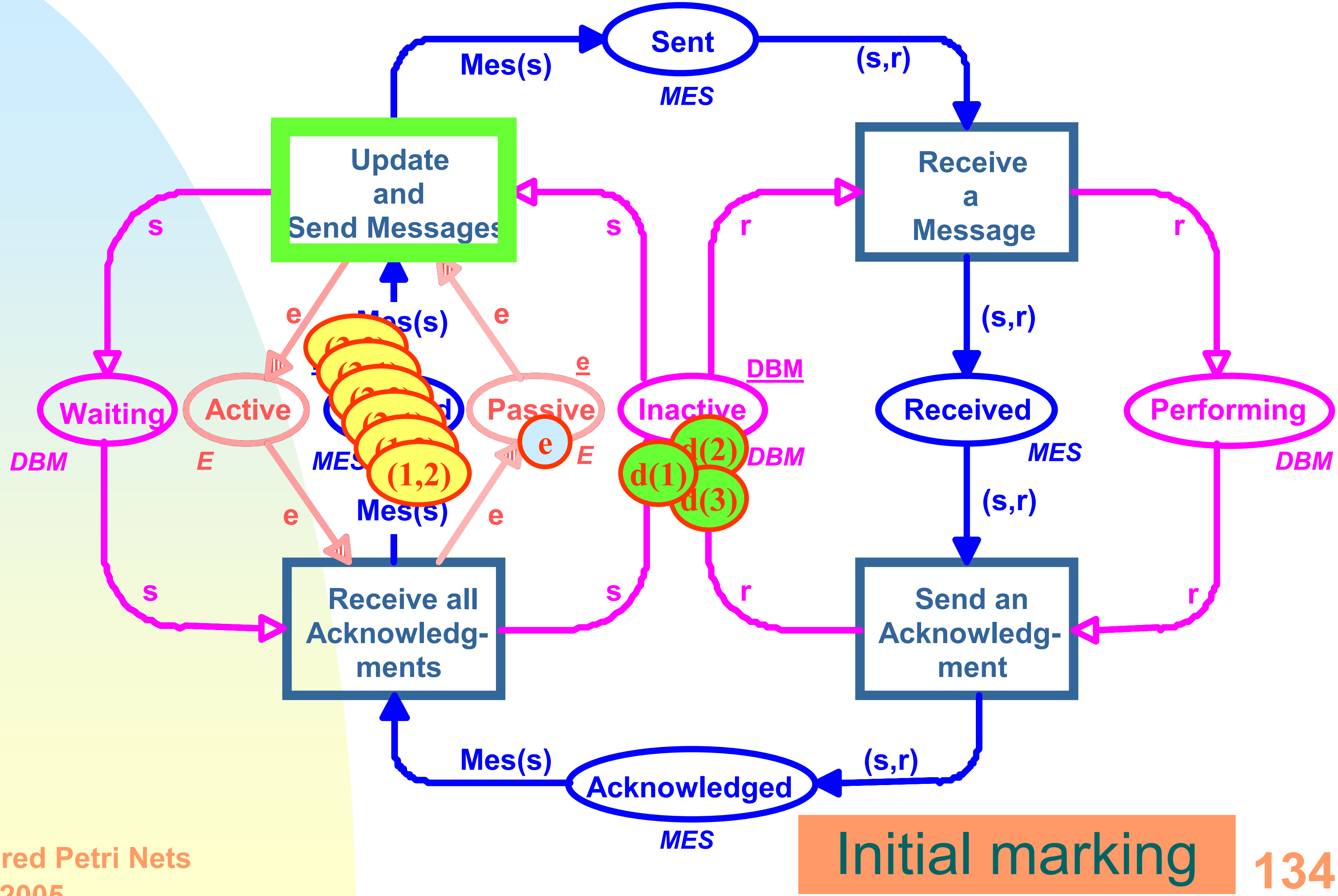
s = d(3)
r = d(2)

Distributed data base



s = d(3)

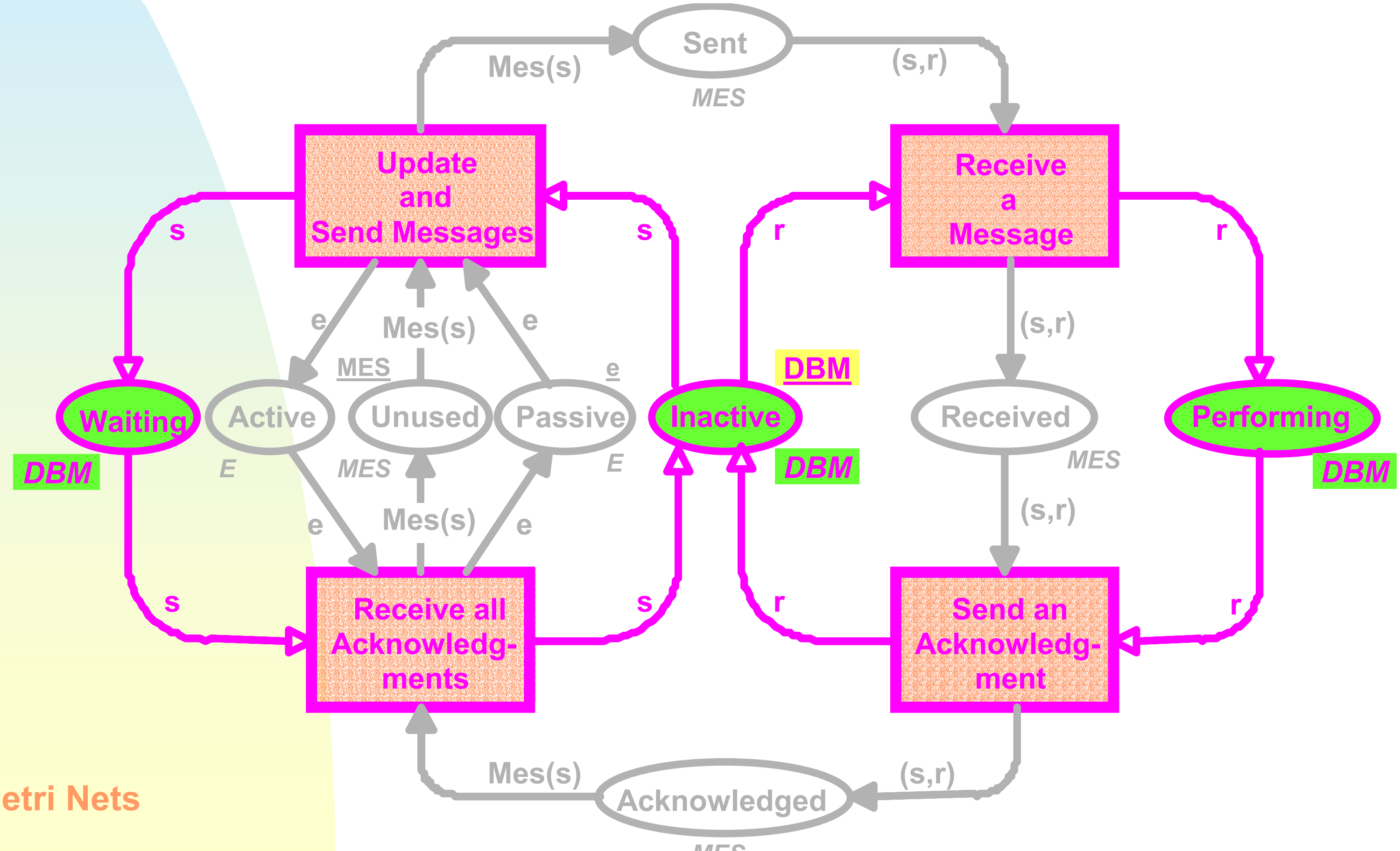
Distributed data base



Initial marking

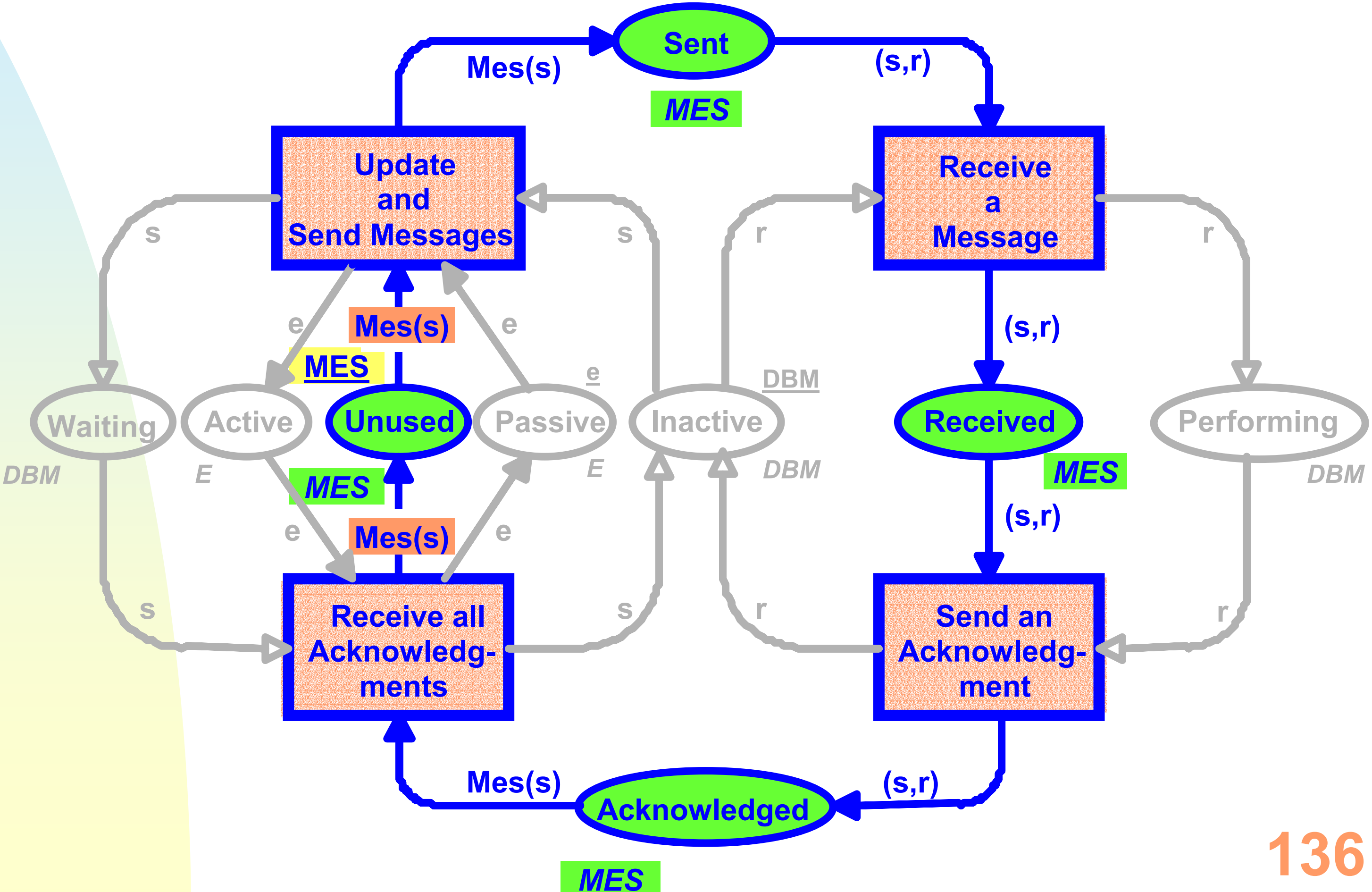
Data base managers

M(Waiting) + M (Inactive) + M(Performing) = DBM



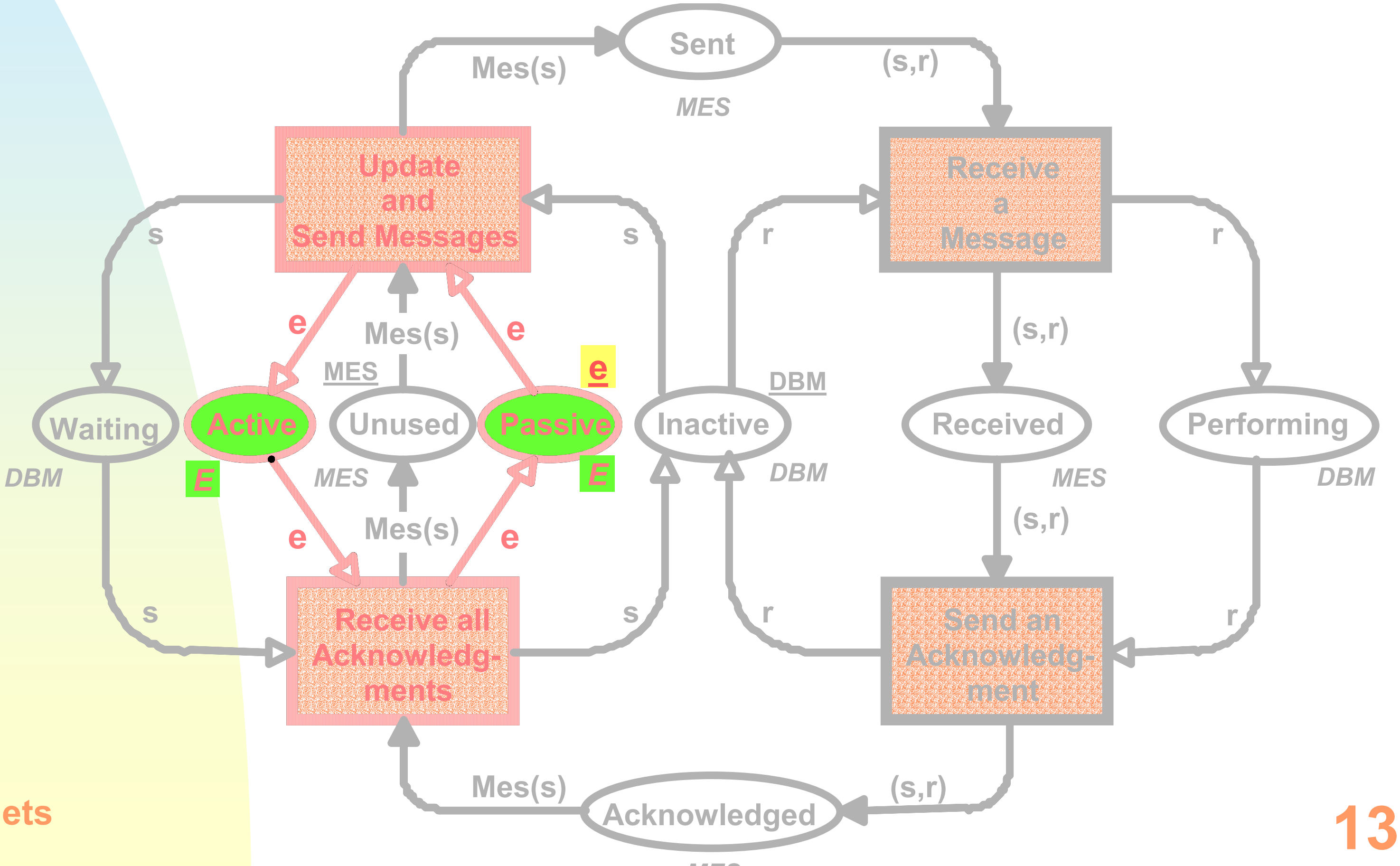
Message buffers

M(Unused) + M(Sent) + M (Received) + M(Acknowl) = MES



Mutual exclusion

M(Active) + M (Passive) = E



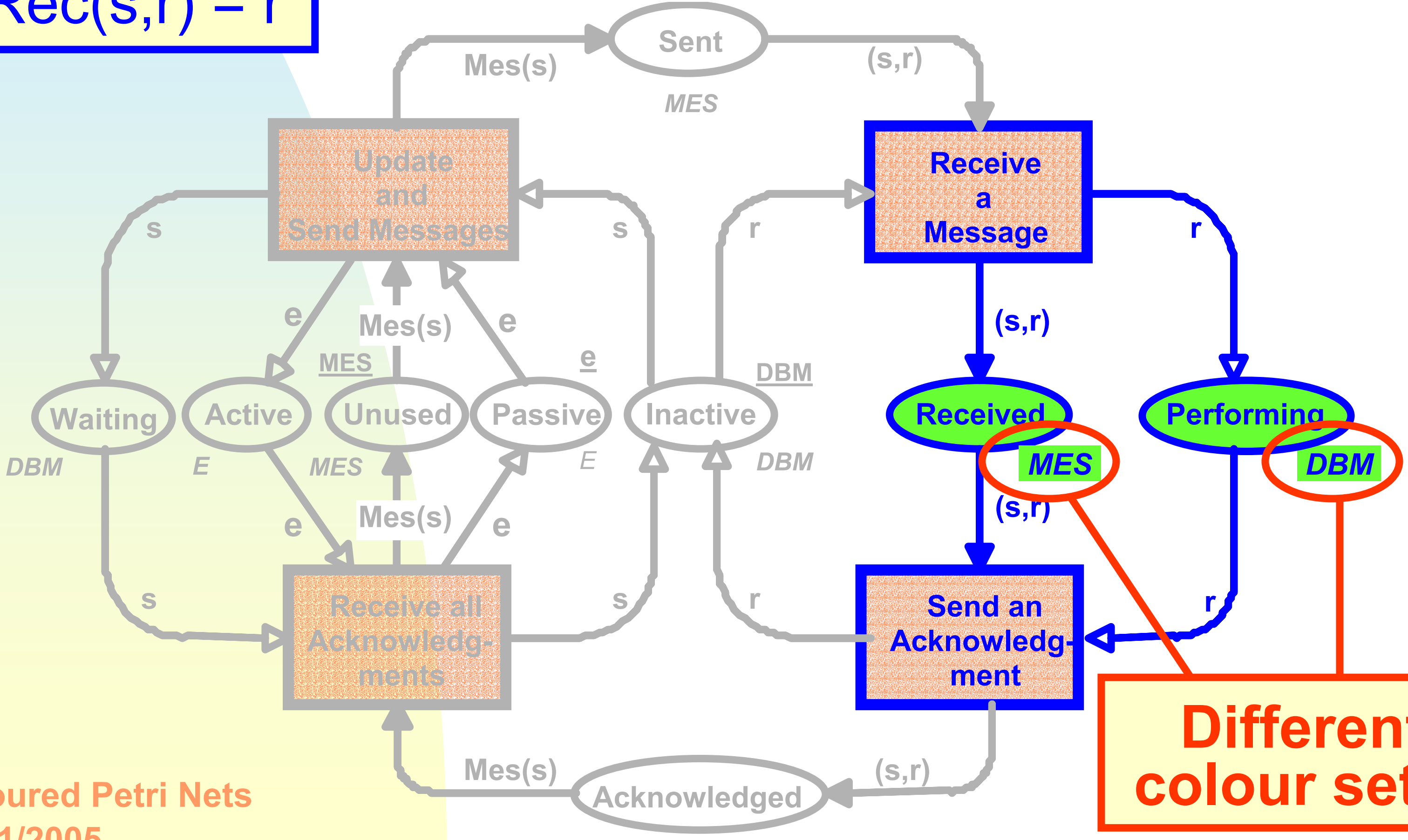
Received messages

Weight function

MES → DBM

Rec(s,r) = r

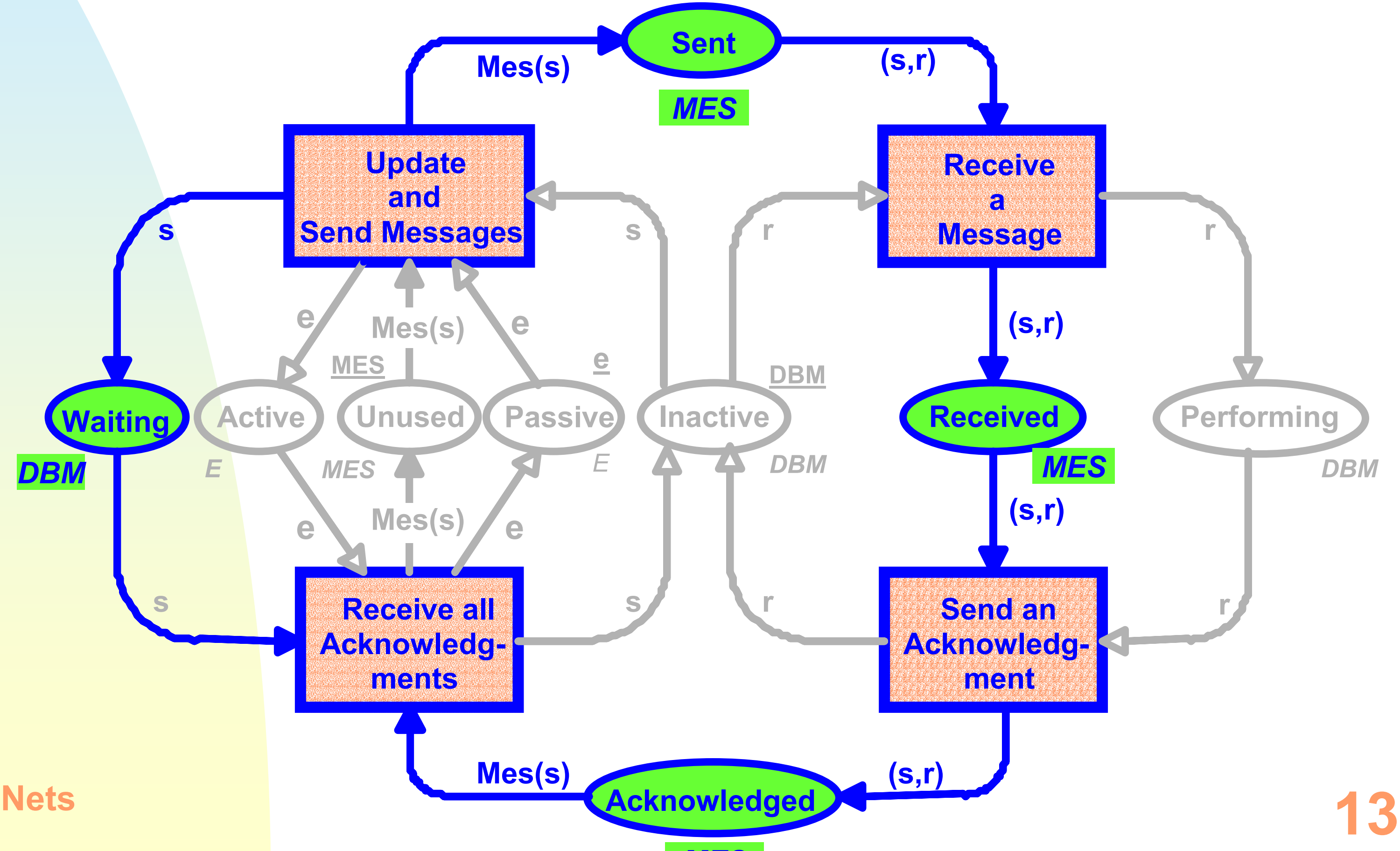
Rec(M(Received)) = M(Performing)



Different colour sets!

Used messages

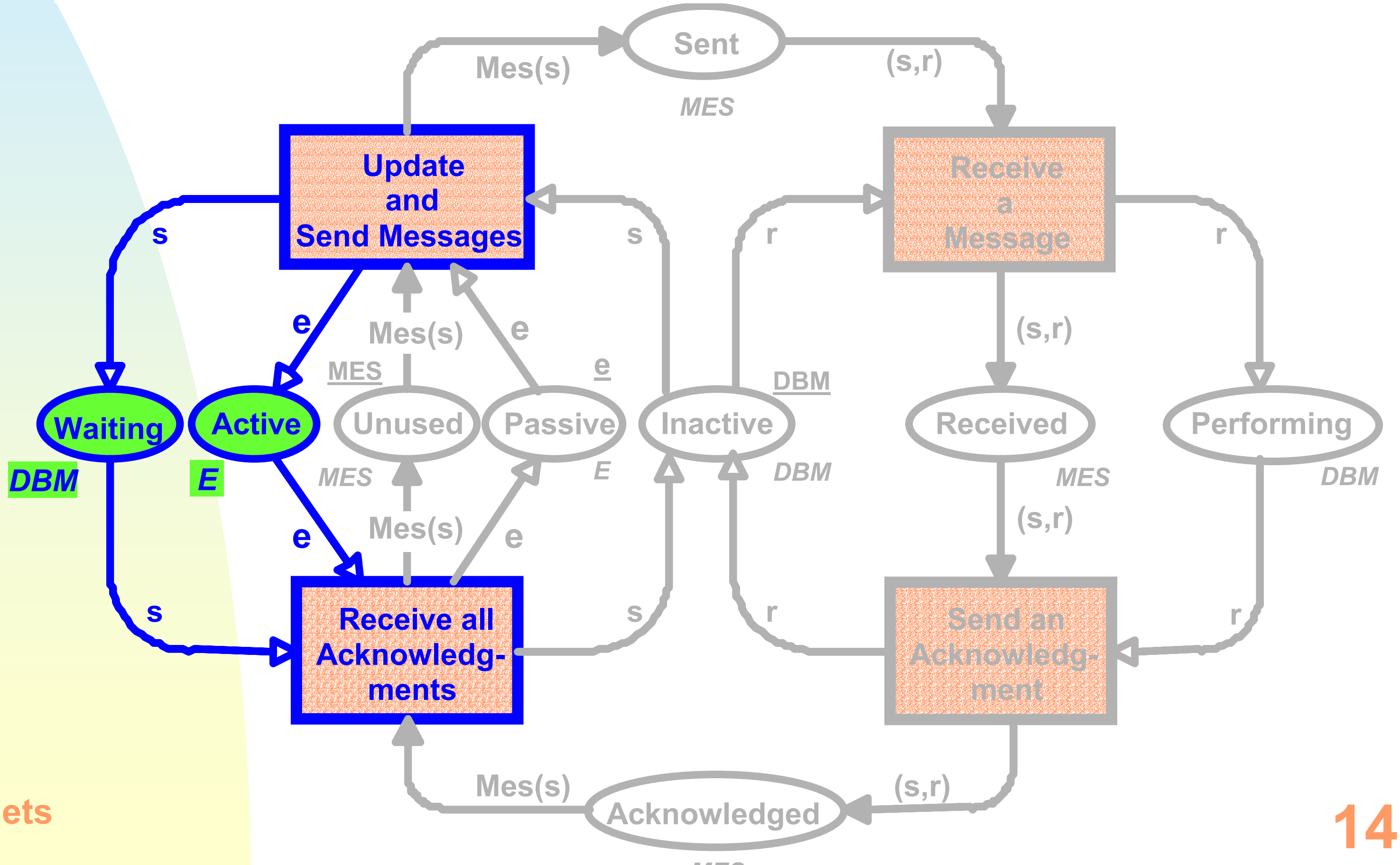
Mes(Waiting) = M(Sent) + M (Received) + M(Acknowledged)



Active and waiting

Ign(M(Waiting)) = M (Active)

Ign(x) = e



Place invariants

Place → M(Place)

- Waiting+ Inactive + Performing = DBM
- Unused + Sent +Receive + Acknowledged = MES
- Active + Passive = E
- *Rec*(Received) = Performing
- *Mes*(Waiting) = Sent + Received + Acknowledge
- *Ign*(Waiting) = Active

More invariants can be obtained by *linear combinations:*

- *Ign*(Waiting) + Passive = E

Construction of invariants

- *Construction* of invariants can be *manual*. This is often straightforward:
  - *System specification.*
  - *Knowledge of system.*
- *Automatic calculation* of *all* place invariants is possible, but:
  - *Rather complex.*
  - *Results* are *difficult* to represent in a form which is *useful for analysis.*
- *Interactive calculation* is much more suitable:
  - The *user* proposes *some* of the weights.
  - The *tool* calculates the *remaining* weights or reports an *inconsistency*.

How to use invariants

- Ordinary *programming languages*:
    - No one would construct a *large program* and then expect *afterwards* to be able to *calculate invariants*.
    - Instead *invariants* are constructed *together* with the program.
- For *CP-nets* we should do the same:
    - During the system specification and modelling the designer gets a lot of *knowledge* about the system.
    - Some of this knowledge can easily be *formulated* as place *invariants*.
    - The *invariants* can be *checked* and in this way *errors* can be found.
    - The *errors* can easily be *located*.

We use invariants to prove behavioural properties of the system

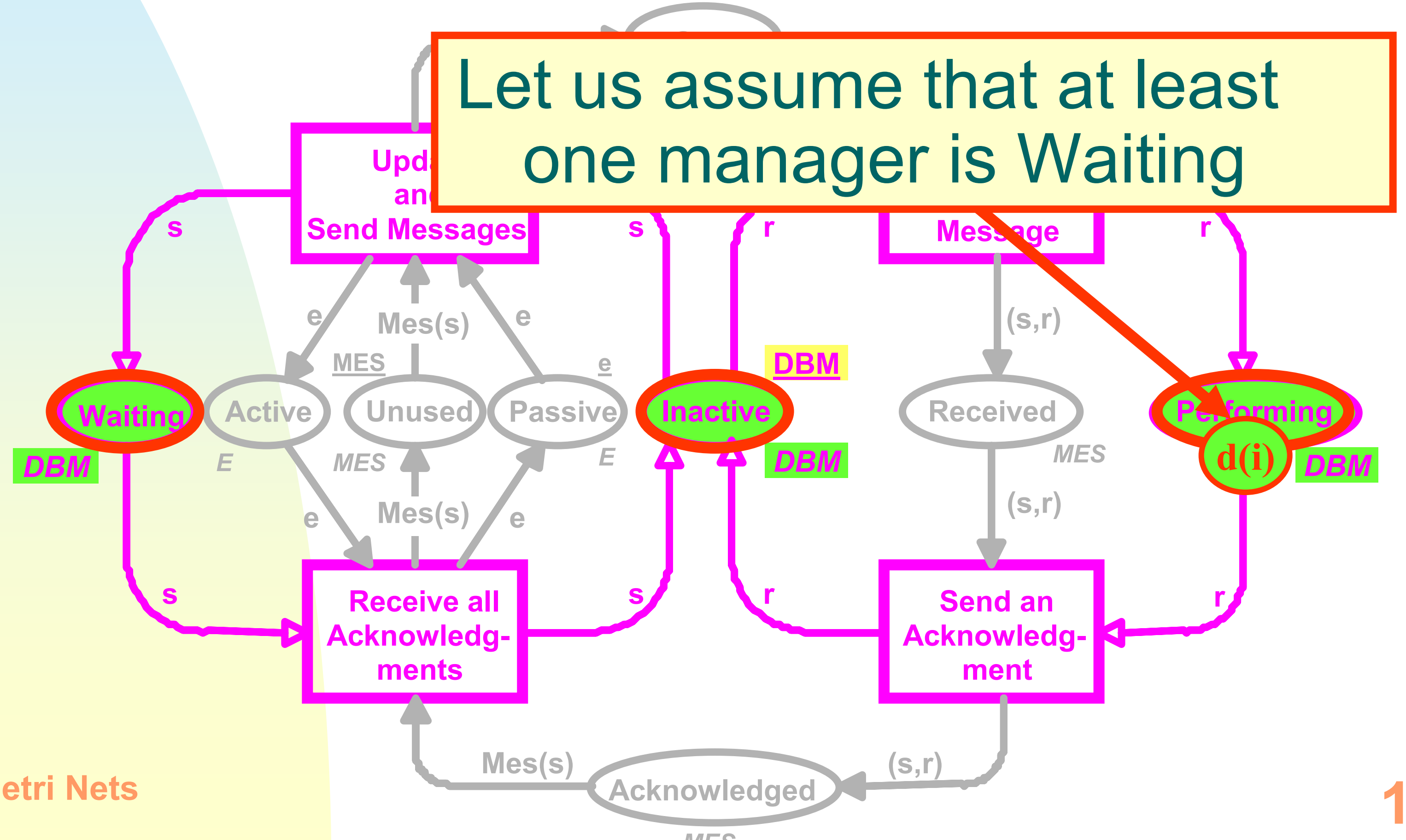
- As an example, let us prove that the data base system *cannot deadlock.*
  - Let a *reachable marking* be given.
  - We will then *prove* that *at least one transition is enabled.*

All invariants are fulfilled

M(Waiting) + M (Inactive) + M(Performing) = DBM

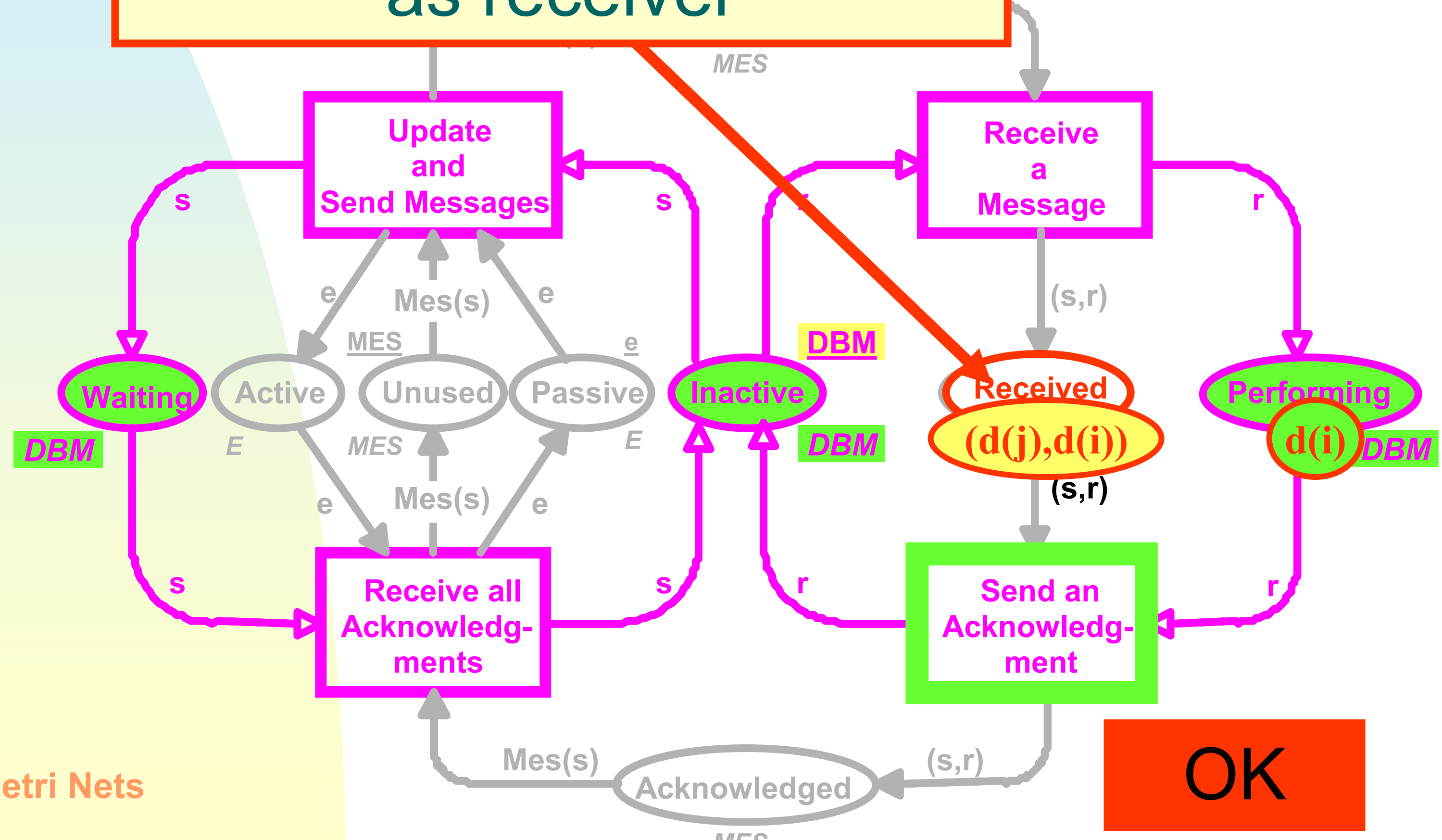
All data base managers must be:

Let us assume that at least one manager is Waiting



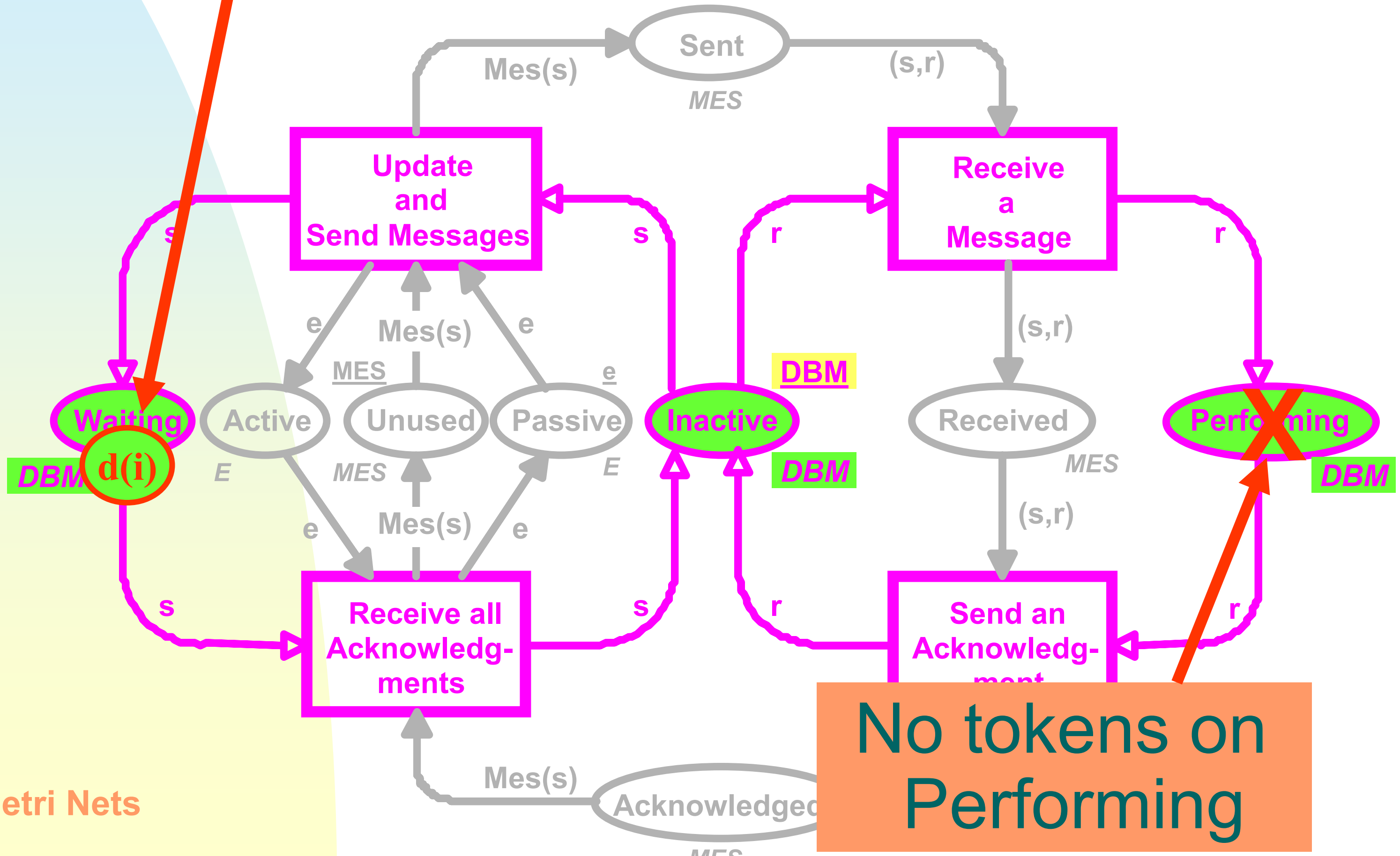
Rec(M(Received)) = M(Performing)

There is a message buffer at Received with d(i) as receiver



OK

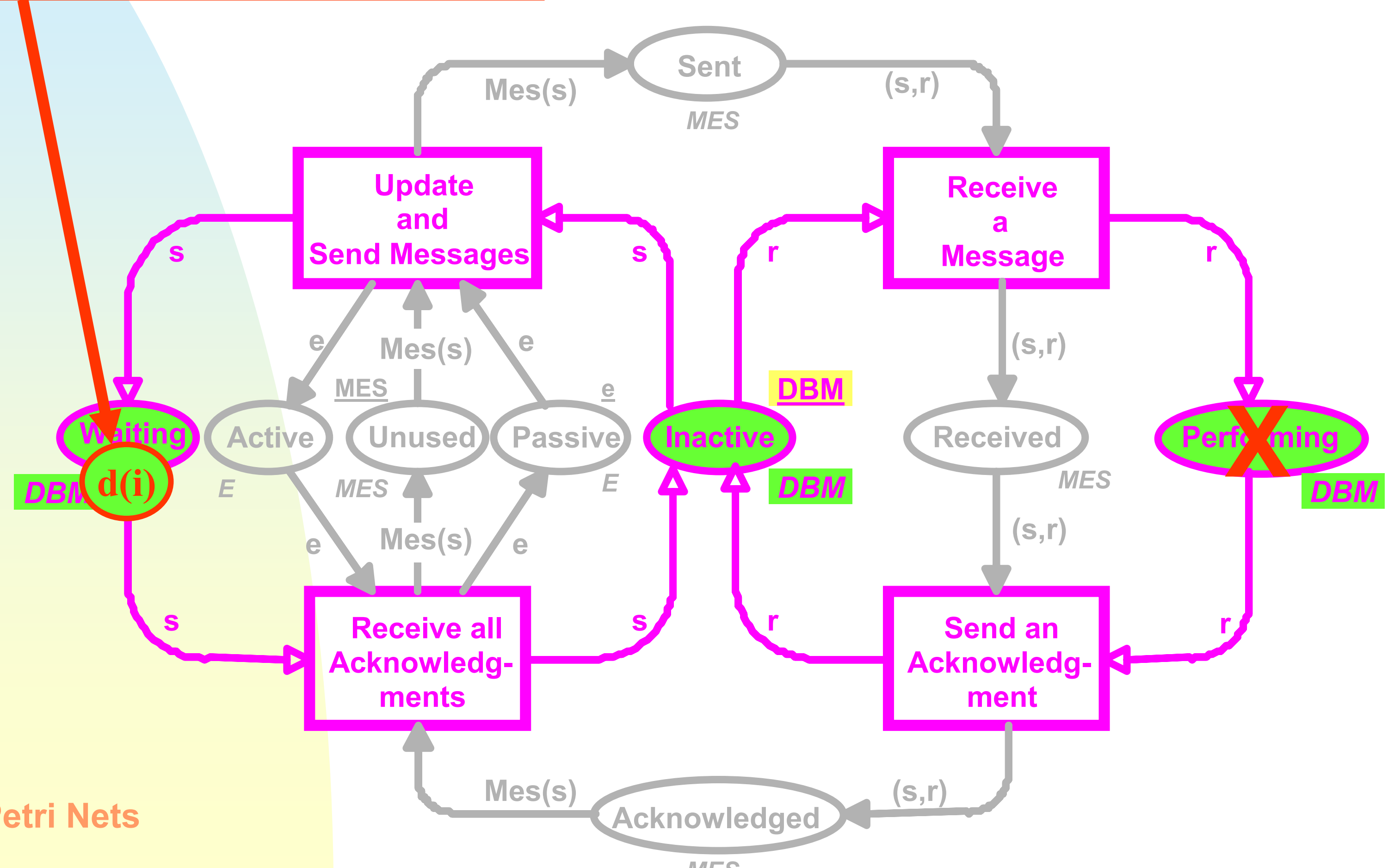
Next let us assume that at least one manager is Waiting



No tokens on Performing

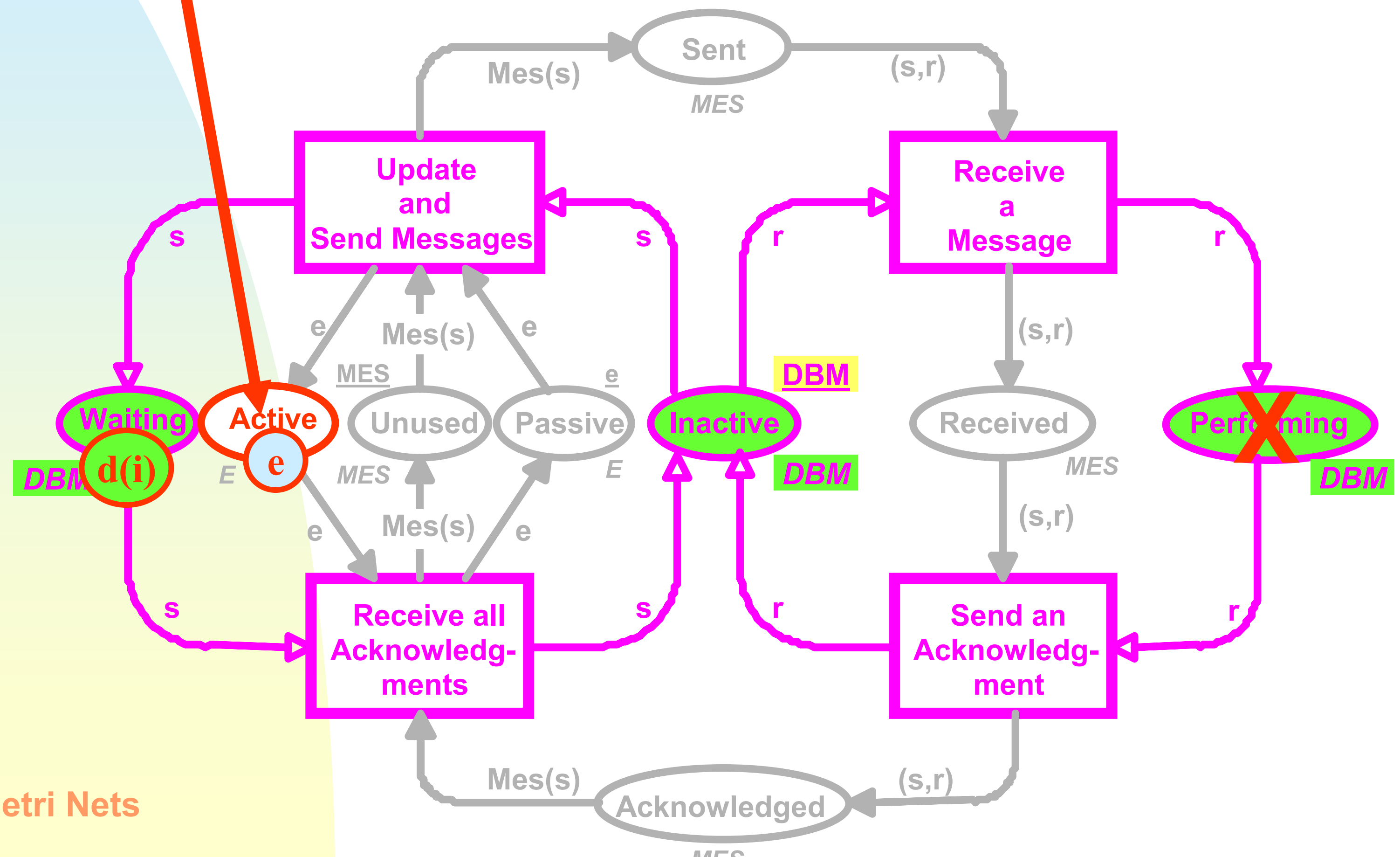
Ign(Waiting) + Passive = E

Exactly one token on Waiting



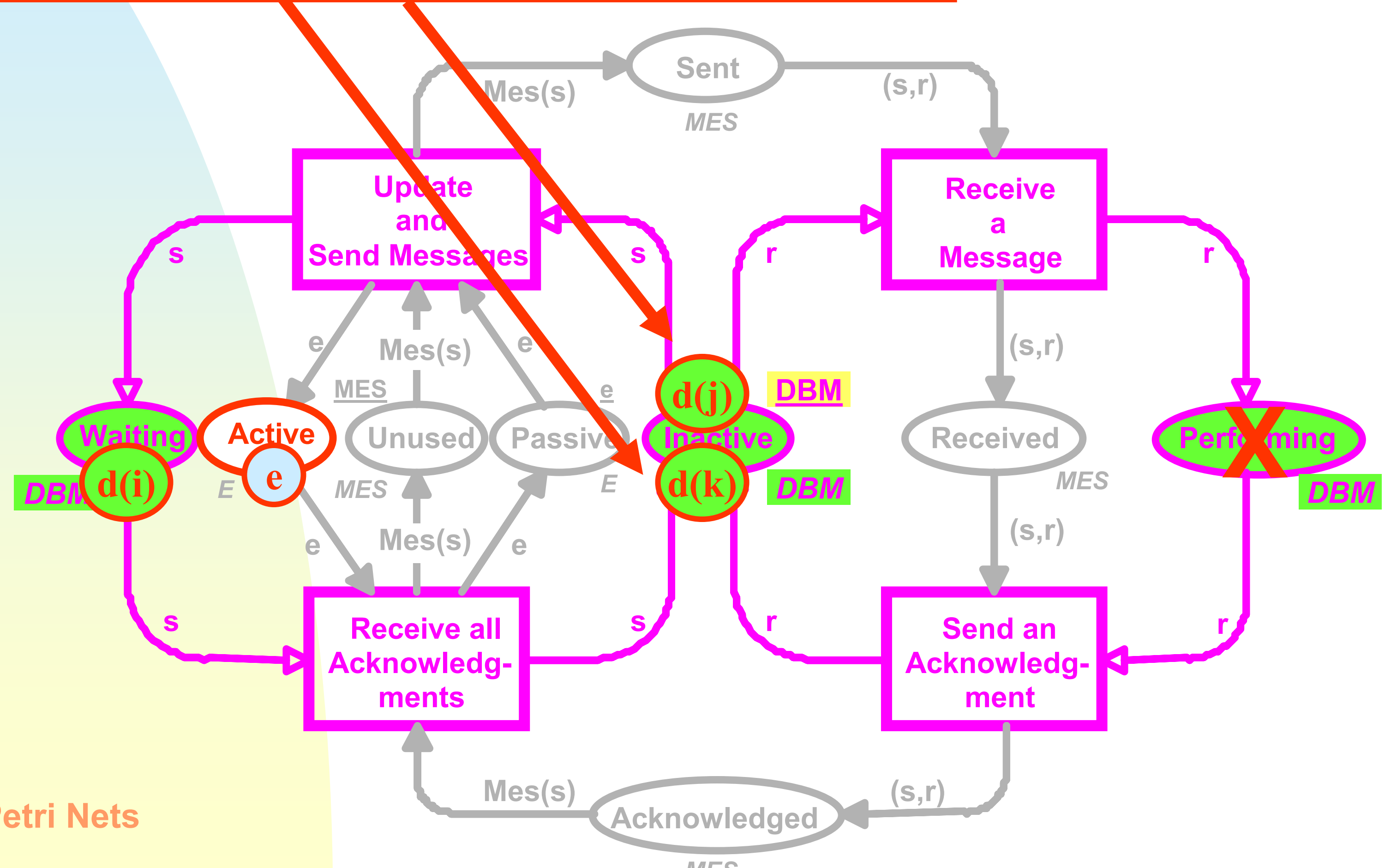
Ign(Waiting) = Active

Exactly one token on Active



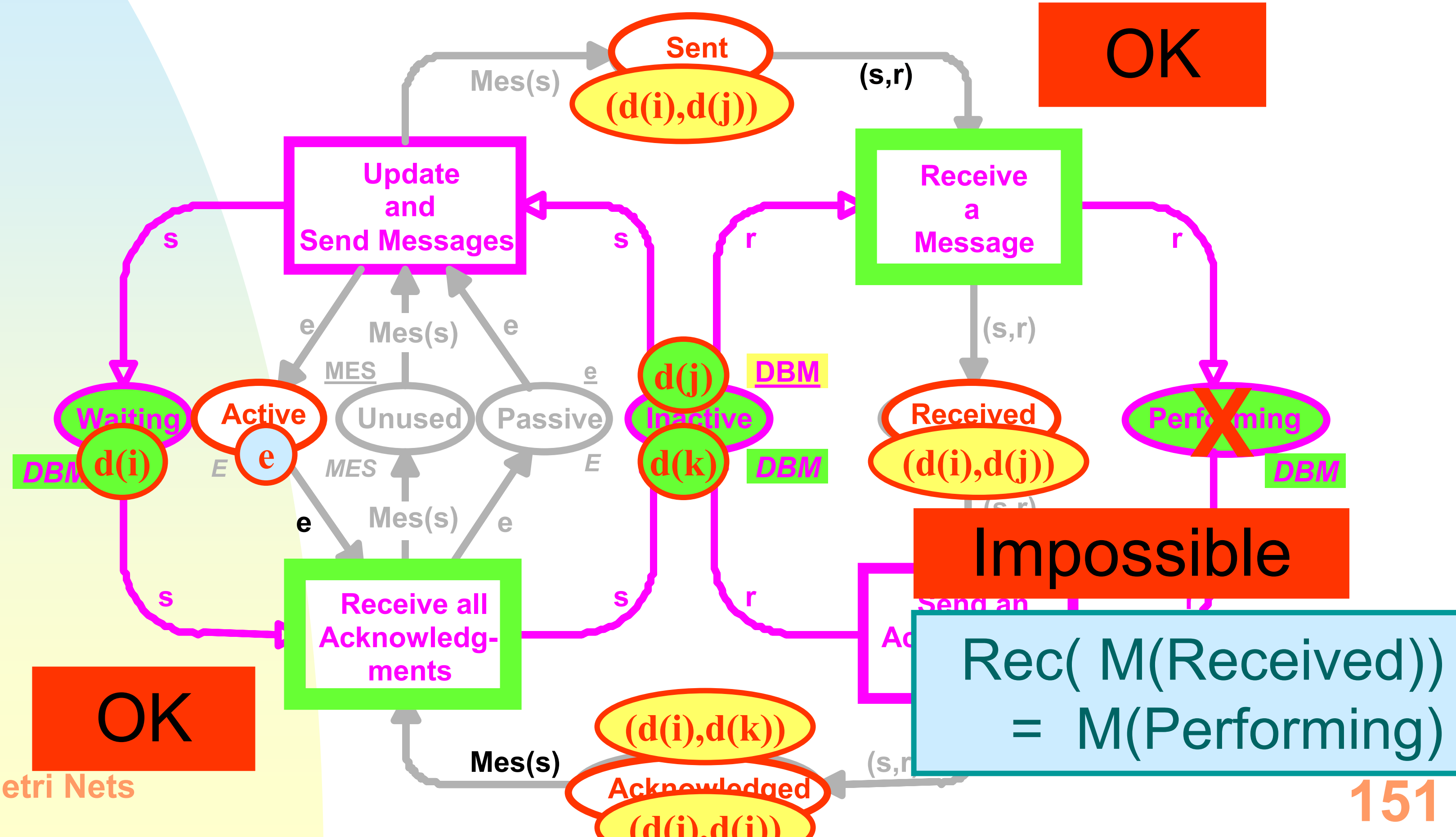
M(Waiting) + M (Inactive) + M(Performing) = DBM

The other data base managers must be Inactive



Mes(Waiting) = M(Sent) + M (Received) + M(Acknowledged)

The message buffers sent by d(i) must be:



OK

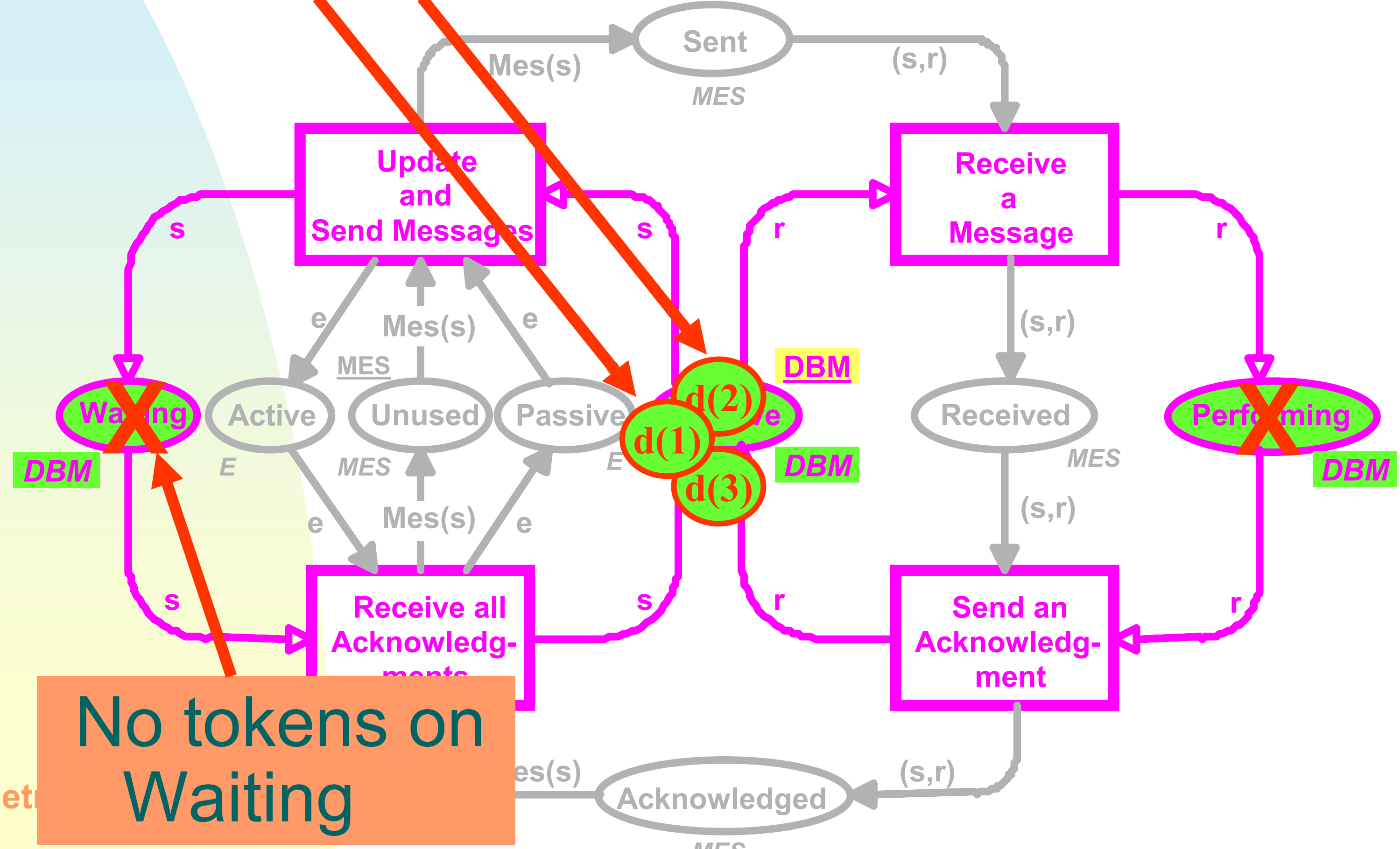
Impossible

OK

Rec(M(Received)) = M(Performing)

M(Waiting) + M (Inactive) + M(Performing) = DBM

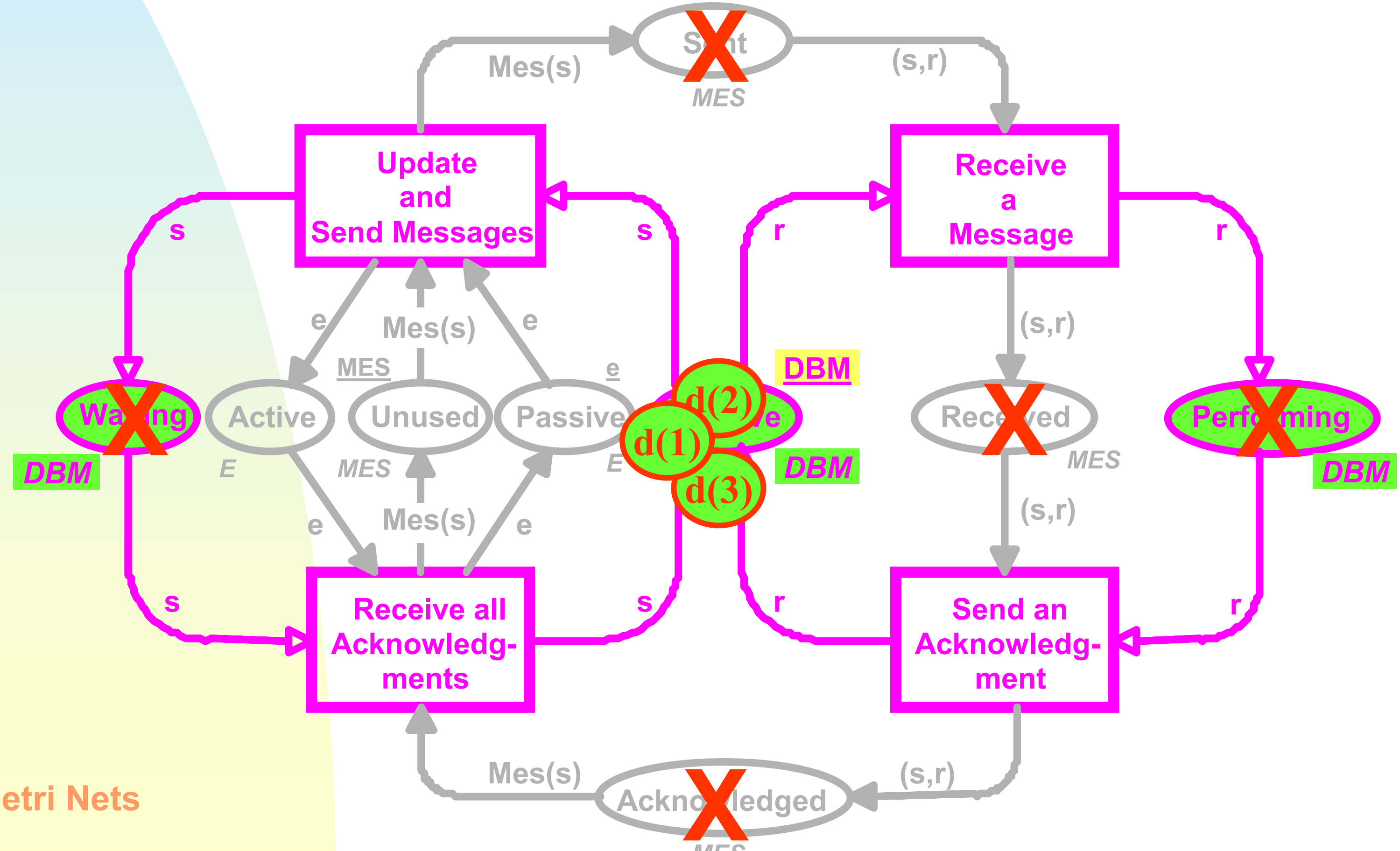
All data base managers must be Inactive



No tokens on Waiting

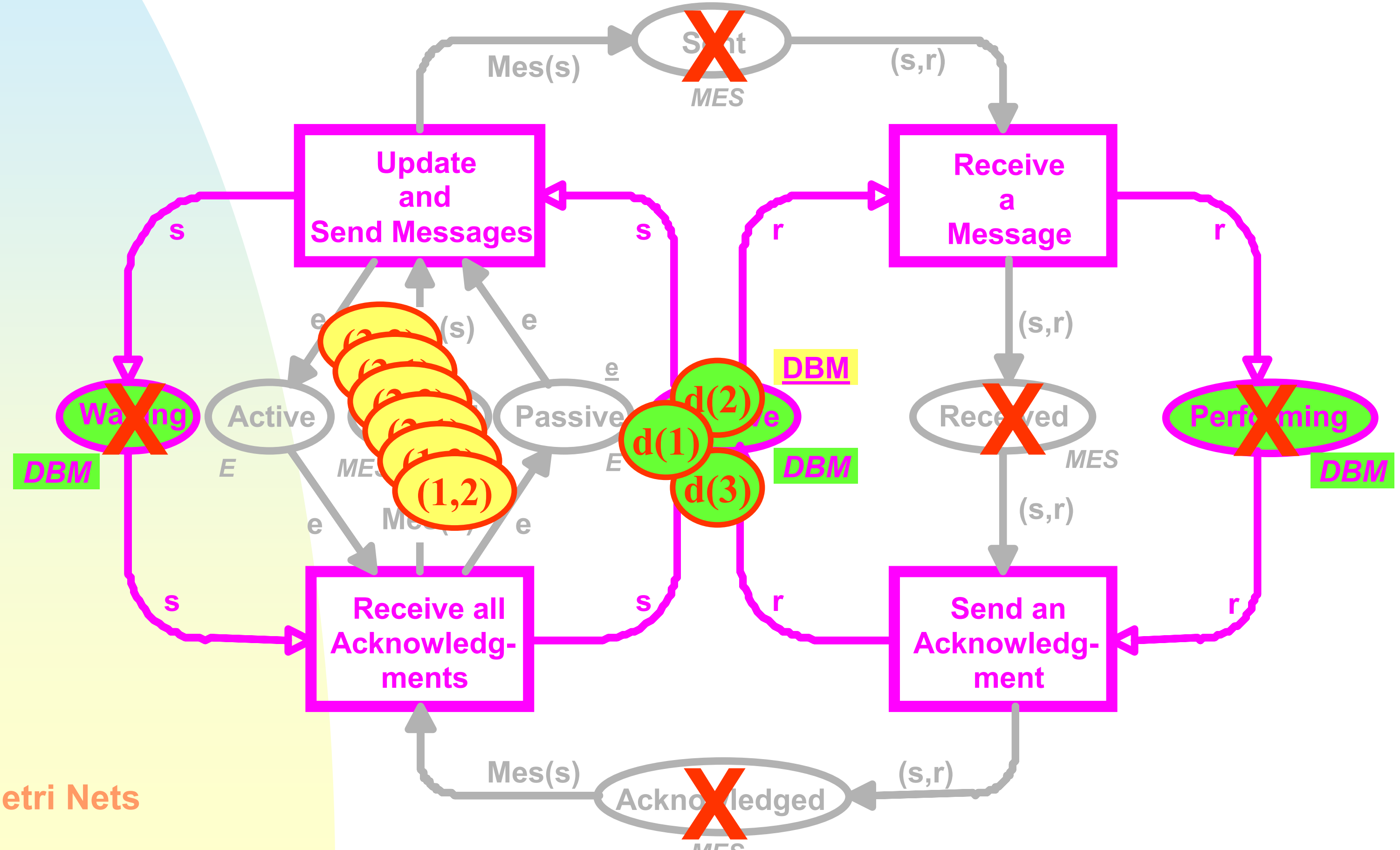
Mes(Waiting) = M(Sent) + M (Received) + M(Acknowledged)

No tokens on Sent, Received, and Acknowledged



M(Unused) + M(Sent) + M (Received) + M(Acknowl) = MES

All message buffers are Unused



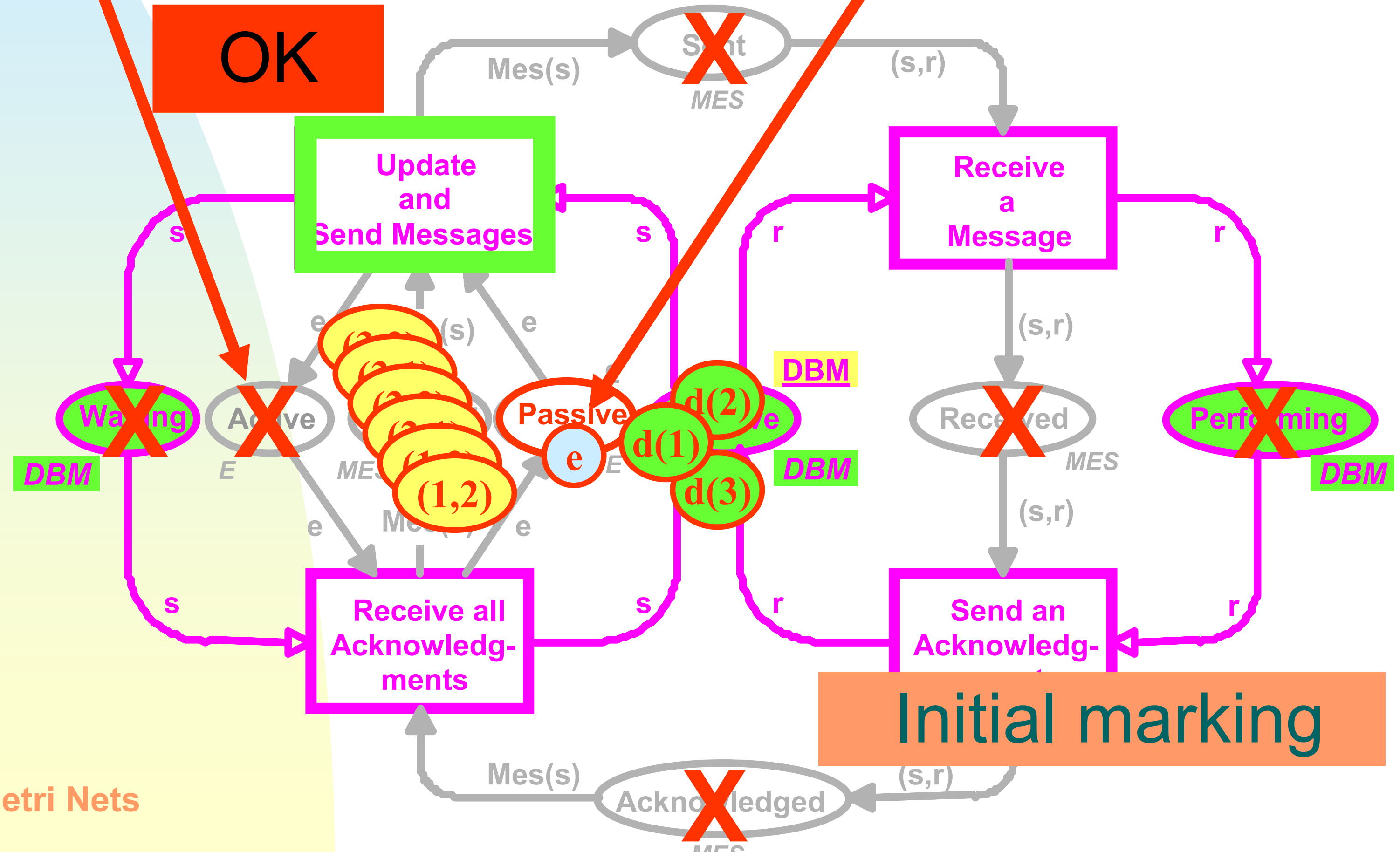
Ign(Waiting) = Active

Active + Passive = E

No tokens on Active

One e-token on Passive

OK



Initial marking

We have now investigated all possible reachable markings

- For each of them we have used the *invariants* to *prove* that *at least one transition is enabled*.
- Hence, we *conclude* that the data base system *cannot deadlock.*

Invariants - pro/contra

- Invariants can be used to verify *large systems.*
  - *No complexity* problems.
  - It is possible to *combine* invariants from *individual modules*.
- Invariants can be used to verify a system *without fixing* the *system parameters* such as the number of sites in the data base system.
- The main drawback is that the *user* needs some *ingenuity* to:
  - *Construct invariants.* This can be supported by *computer tools – interactive process.*
  - *Use invariants.* This can also be supported by *computer tools – interactive process.*

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.*

More information on CP-nets

- The following *web-pages* contain a lot of information about CP-nets and their tools:

  http://www.daimi.au.dk/CPnets/

- *Introduction to CP-nets*, including a number of detailed examples.
- Manual for *CPN Tools*.
  - The tool is *free of charge* also for commercial companies.
- A list of more than 50 published papers describing different *industrial applications* of CP-nets and the CPN tools.
- Details of a 3-volume *CPN text book*.