Introduction to SDL

TSP

Stéphane Maag

TELECOM SudParis

Specification Description Language

Outline

- SDL, a FDT for complex system specification
- MSC to SDL
- SDL system
- SDL notations
- SDL process
- From the specification to the simulation
- RTDS

... and conclusion.

SDL - a Formal Description Technique

- FDTs (also called *specification language*):
  - specify the functional properties of a system according to its environment
  - are conceived to describe distributed systems composed by processes that are executed in parallel, synchronize themselves and communicate by messages
- Other techniques: process algebra (CCS), finite state machines, temporal logic, Petri networks, ...

Briefly, SDL

- SDL (Specification Description Language):
  - Define and normalized by ITU(-T) (1988, 1992, 1996, 2000)
  - based on the Extended Finite State Machines (EFSM), asynchronous
  - 2 visions: SDL-GR (graphical) and SDL-PR (textual)
  - Abstract data types, ASN.1

SDL for Reactive and Discrete Systems

- Communication:
  - Message exchanges between the system and its environment
  - Mainly asynchronous interactions, but synchronous ones also supported

- Nevertheless:
  - SDL is not adapted to cyclic data-driven inputs
  - SDL is unable to describe non real-time aspects, such as:
    - Data bases
    - GUIs

SDL applications

- Wide range of applications
  - safety and mission critical communicating systems
  - real-time applications

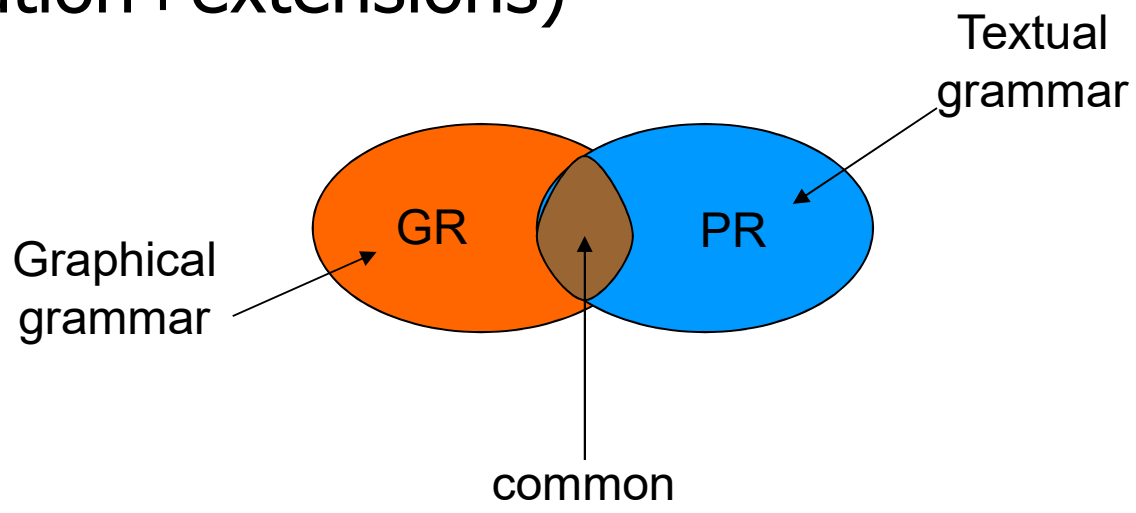
- Wide range of architectures
  - workstation-based distributed system, 32-bits communication board, 8-bits micro-controller embedded system

SDL architecture and behaviors

- To specify, to describe without ambiguities telecommunication systems
- To represent functional properties of a system:
  - structural properties: system architecture, its decomposition into interconnected functional blocks
  - behavioral properties: system reactions after stimuli coming from the environment
- The architecture ≠ The behavior

SDL
Two normalized representations

- Graphical representation: GR
- Textual representation: PR
- Exchange format: PR+CIF (information+extensions)



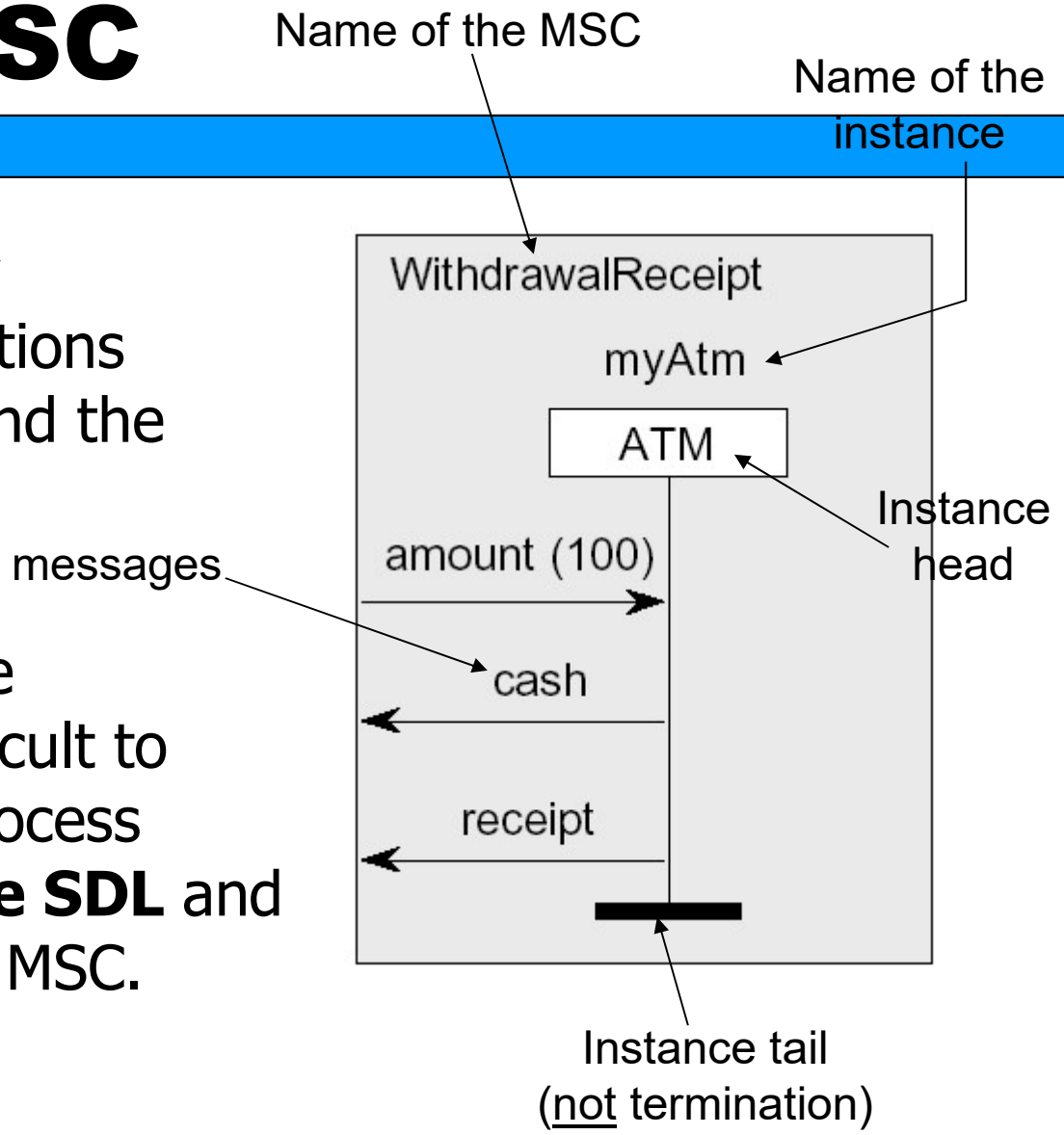
MSC - to provide the behaviors

Message Sequence Chart

- Z.120 Recommendation managed by the ITU
- "*is to provide a trace language for the specification and description of the communication behavior of system components and their environment by means of message interchange*"

SDL with MSC

- To describe cases by sequences of interactions between instances and the environment
- allows to observe the interactions, but difficult to assign values and process operations … **we use SDL** and we may control with MSC.



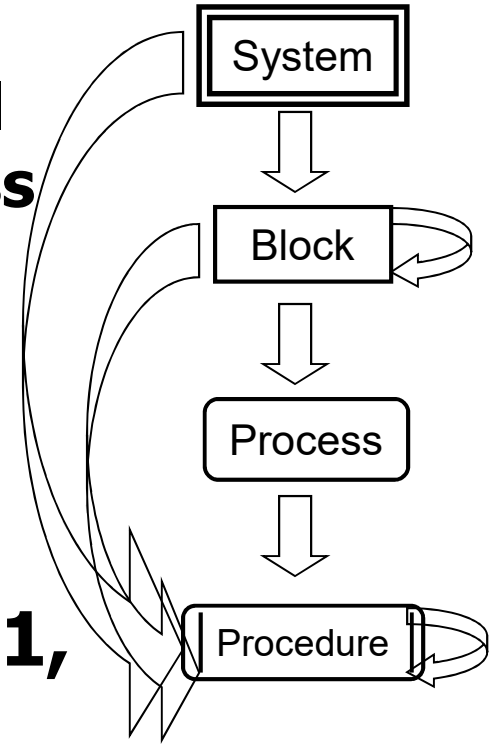
System specification

Three aspects in order to specify:

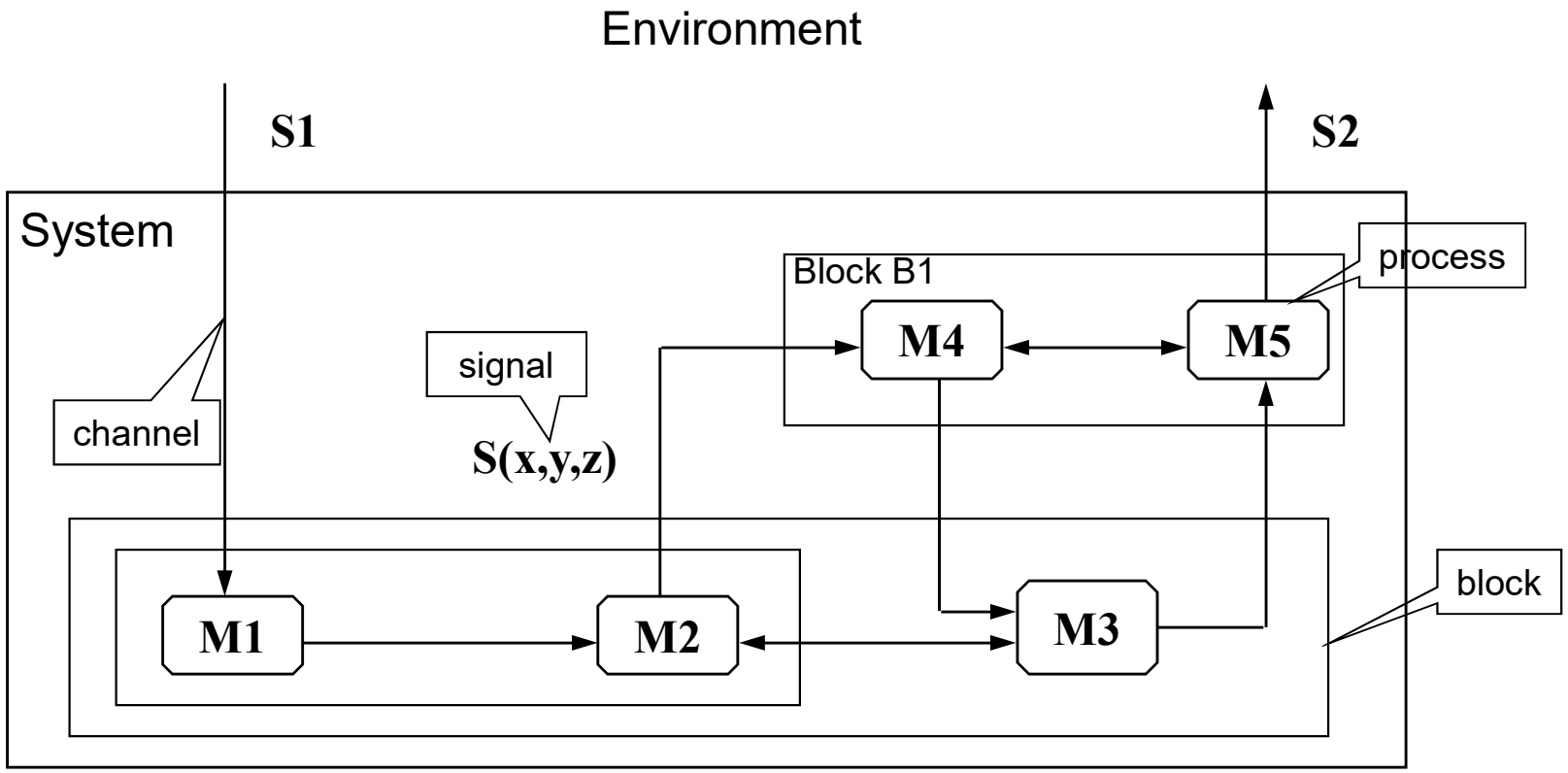
- The definition of the system structure with the interconnections
- The dynamic behavior of each process (or machines) and their interaction with the other processes and the environment
- operations on data (into the processes)

Semantic models - Hierarchy

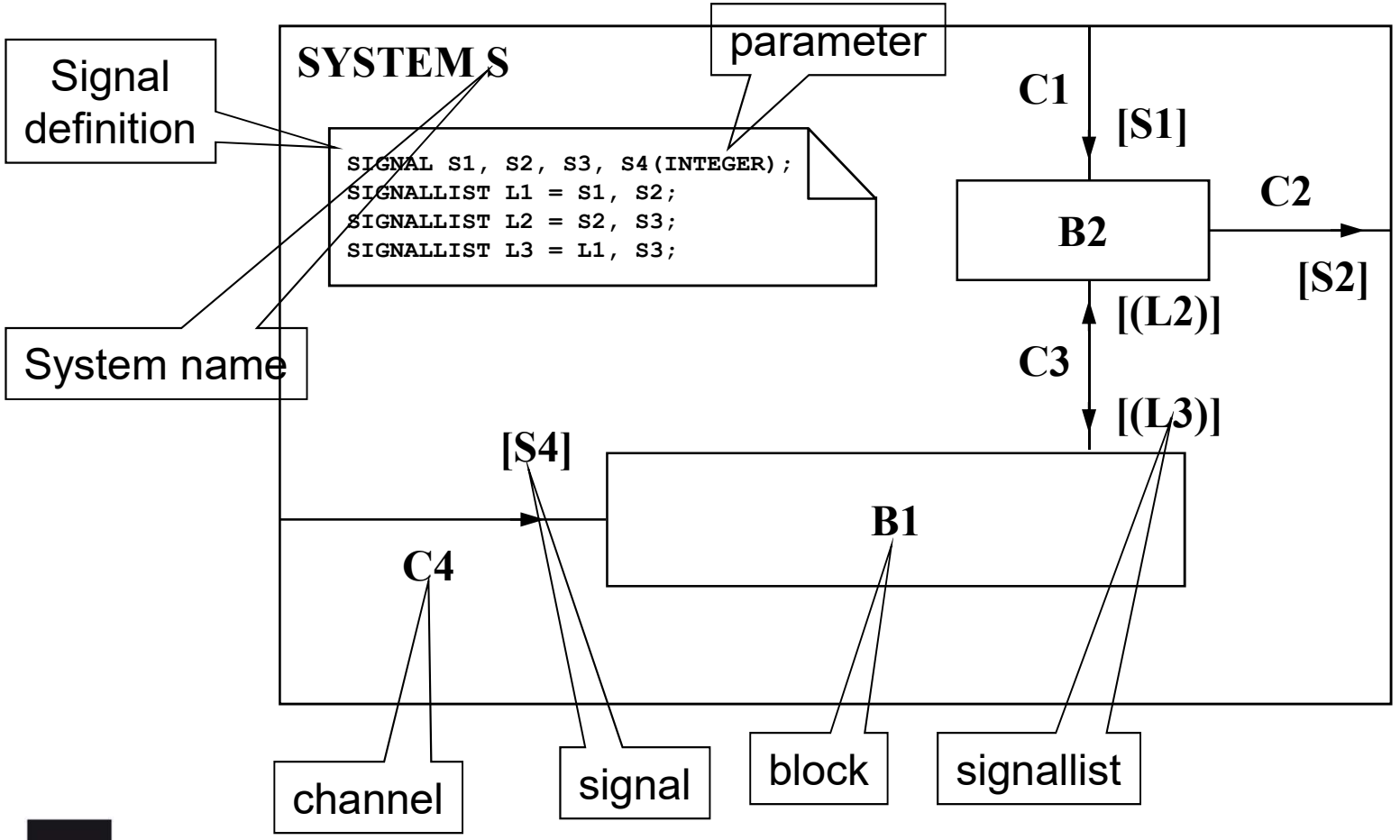
- System architecture:
  - Decomposition by interconnected structural entities: **system**, **block, channel, process**
- System behavior:
  - communicating processes: **signals, variables as inputs/outputs: EFSM**
- Data: **variables, signals, sorts, ASN.1, ...**



System architecture

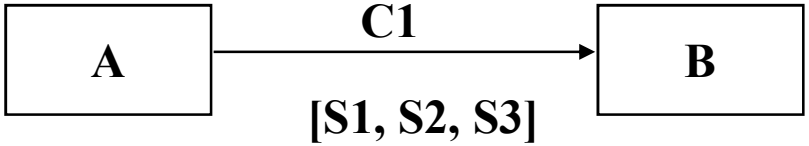


System SDL: example

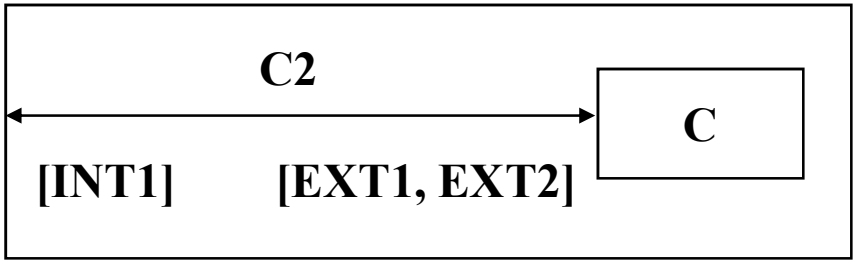


Channels

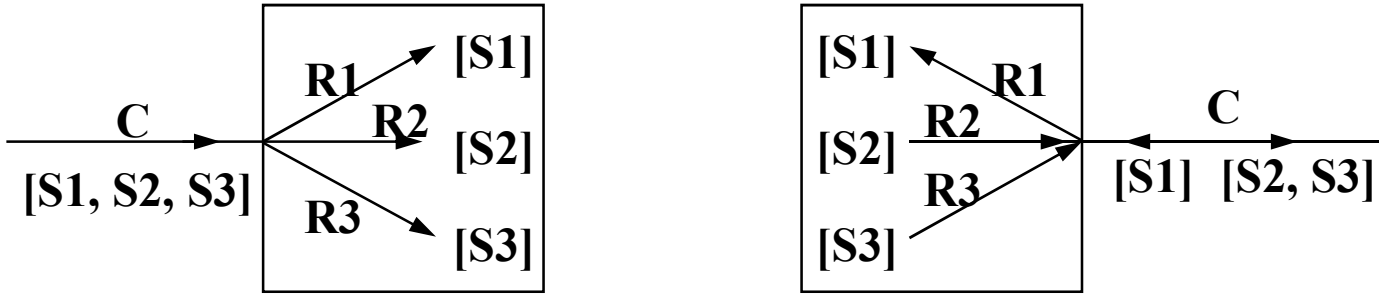
unidirectional



bi-directional



Multi-connections



SDL predefined types

- *INTEGER* signed integer
- *REAL* real
- *NATURAL* positive or null integer
- *CHARACTER* 1 character
- *CHARSTRING* charstring (string of characters)
- *BOOLEAN* boolean
- *TIME* absolute time (syntype of REAL)
- *DURATION* duration (syntype of REAL)
- *PID* to identify a process instance

Operators on predefined types

- All types
  - =, /=
- INTEGER and NATURAL
  - -, +, *, /, >, <, >=, <=, Float, Mod, Rem
- REAL
  - -, +, *, /, >, <, >=, <=, Fix

CONSTANTS

- They can be defined at any level of the SDL hierarchy

SYNONYM maxusers **INTEGER** = 10**;**

Basic user-defined types

- Enumerated types

```
NEWTYPE WeekDay
LITERALS mon, tue, wed, thu, fri, sat, sun;
ENDNEWTYPE;
```

- Range types (often used to index arrays)

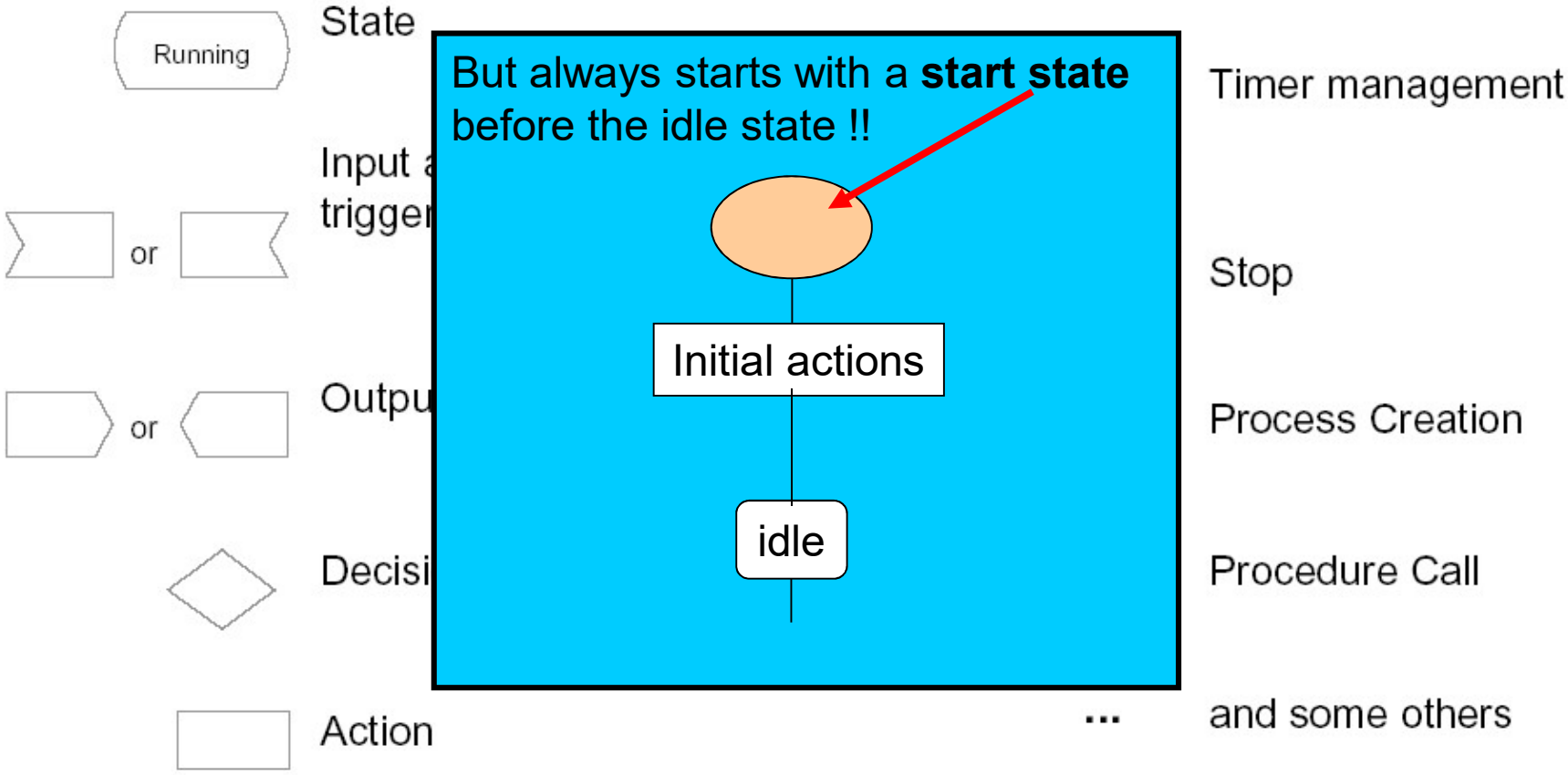
```
SYNTYPE Index_T = Natural
CONSTANTS 1:12
ENDSYNTYPE;

SYNTYPE Digit_T = Character
CONSTANTS '0':'9'
ENDSYNTYPE;

SYNTYPE WeekEnd = WeekDay
DEFAULT sun;  CONSTANTS sat:sun
ENDSYNTYPE;
```

The SDL process

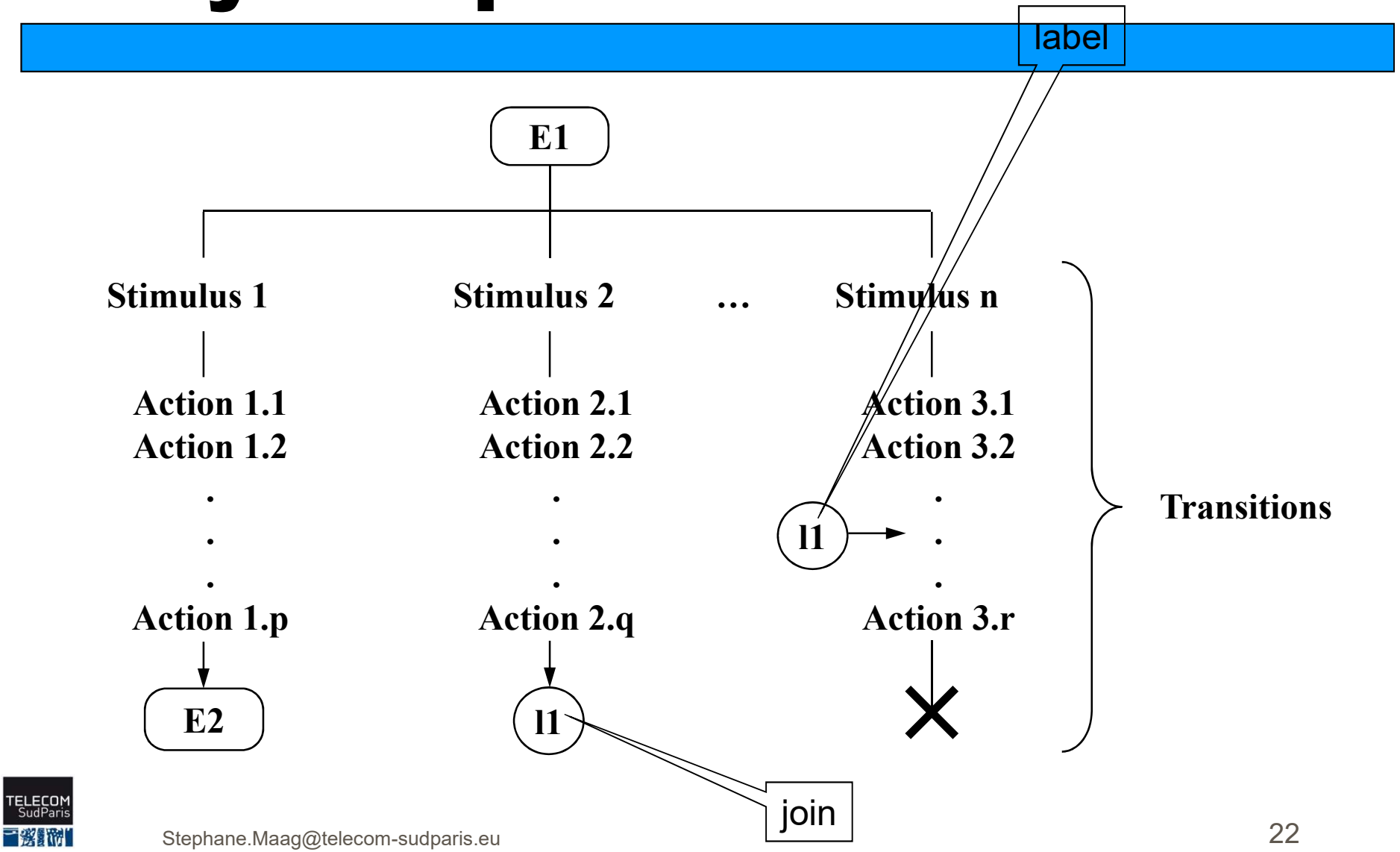
It describes the behavior and extends the FSM concept:

- the queue associated to each process is not necessarily a FIFO.
- A transition (not necessarily of a null length) may contain:
  - receiving and sending data
  - analyzing variables to determine the next transition
  - execution of tasks
  - procedure call
  - dynamic creation of process
  - triggered timers

Major SDL elements in a process



Body of a process

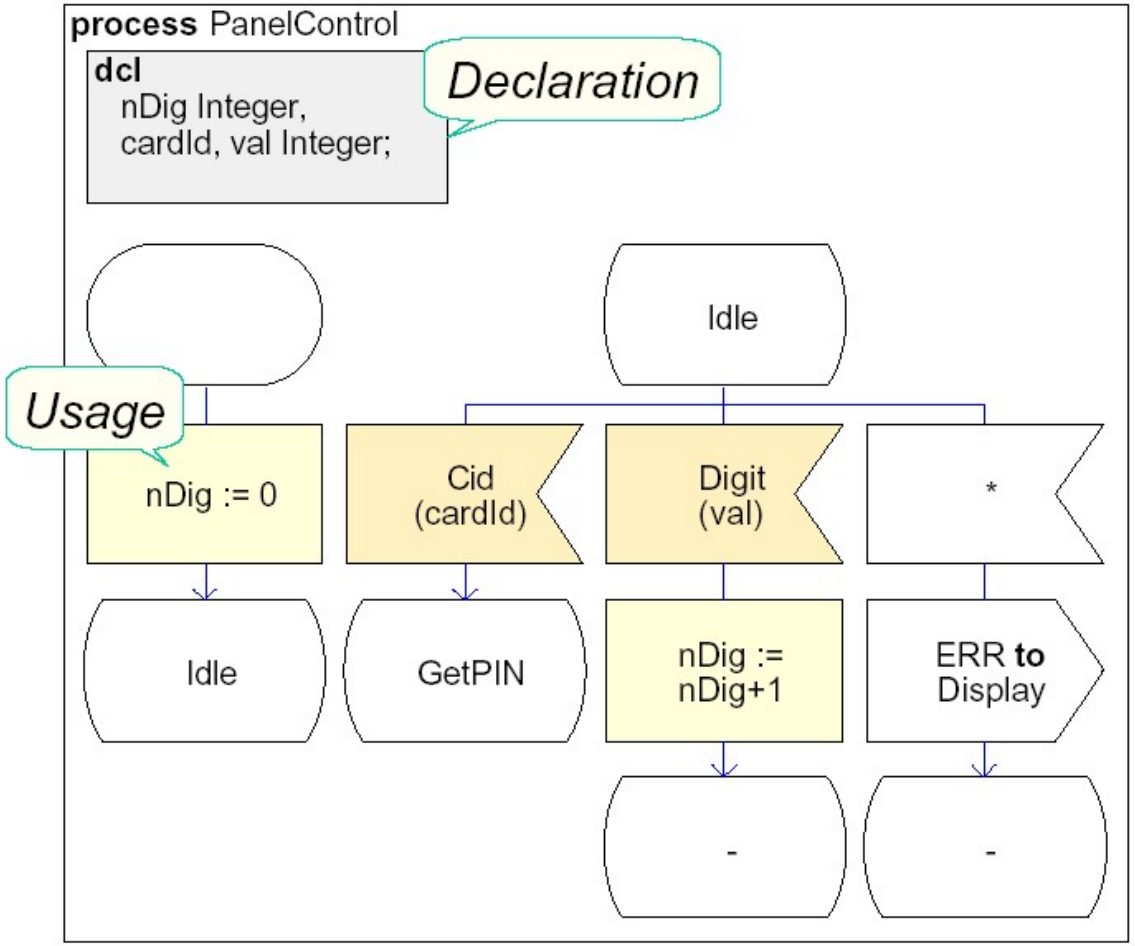


Declaration in processes

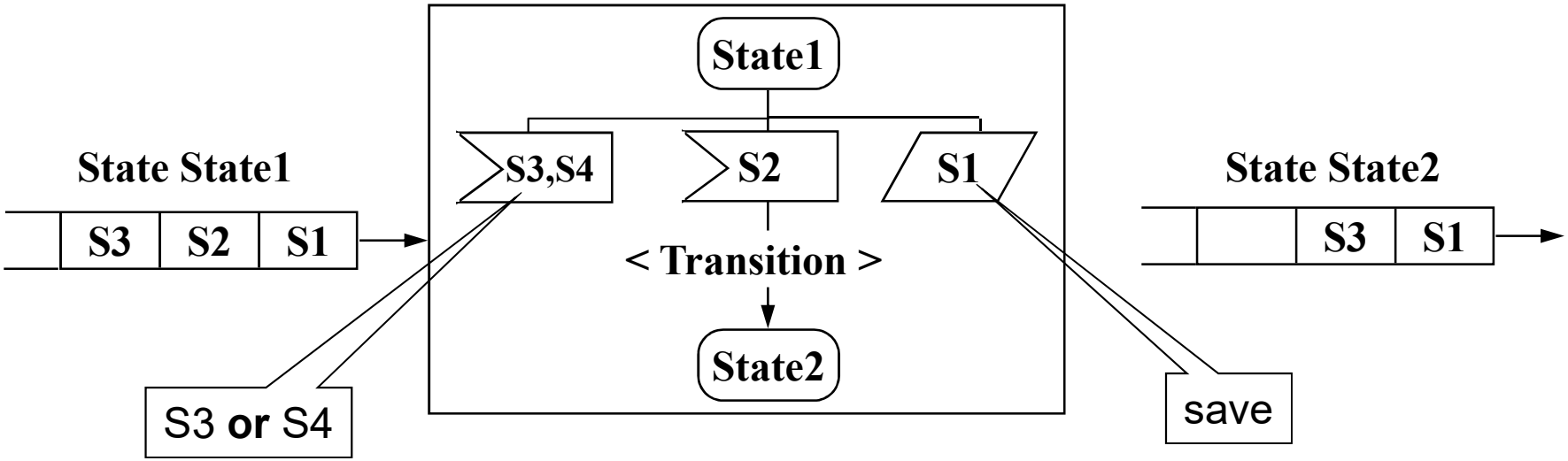
Variables

- declared in a Text symbol of a process, service, procedure
- no global variables at system or block level
- can be initialized:

```
DCL
   nbTransactions Integer := 0,
   v1, v2 MyType;
```

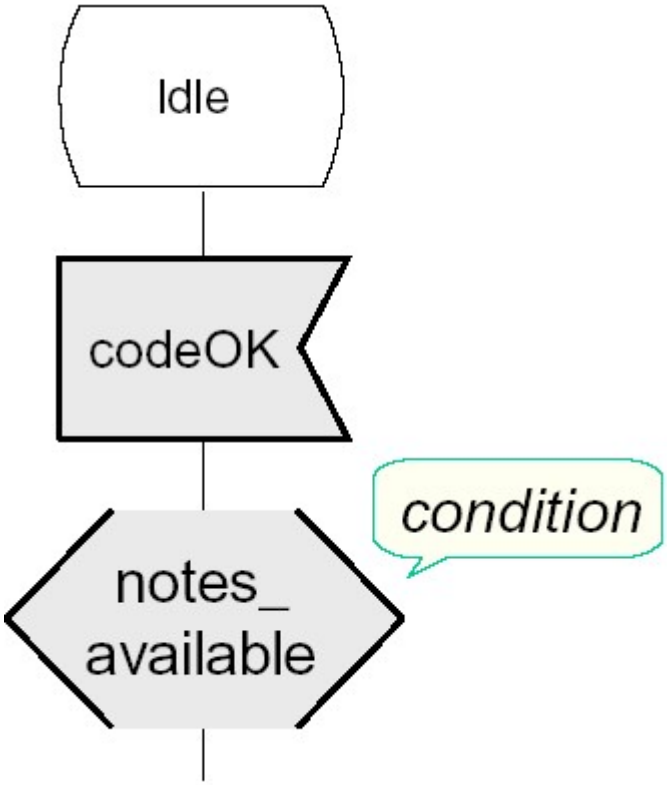


Stimuli types - inputs



"save" allows to save a signal and keeps it in the queue until the next state … waiting for the next signal.

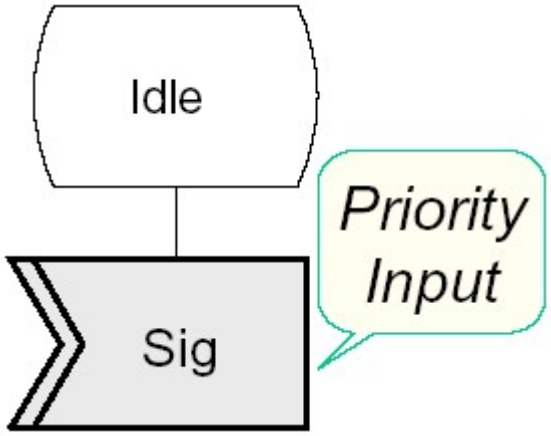
Input - Condition



Boolean expression

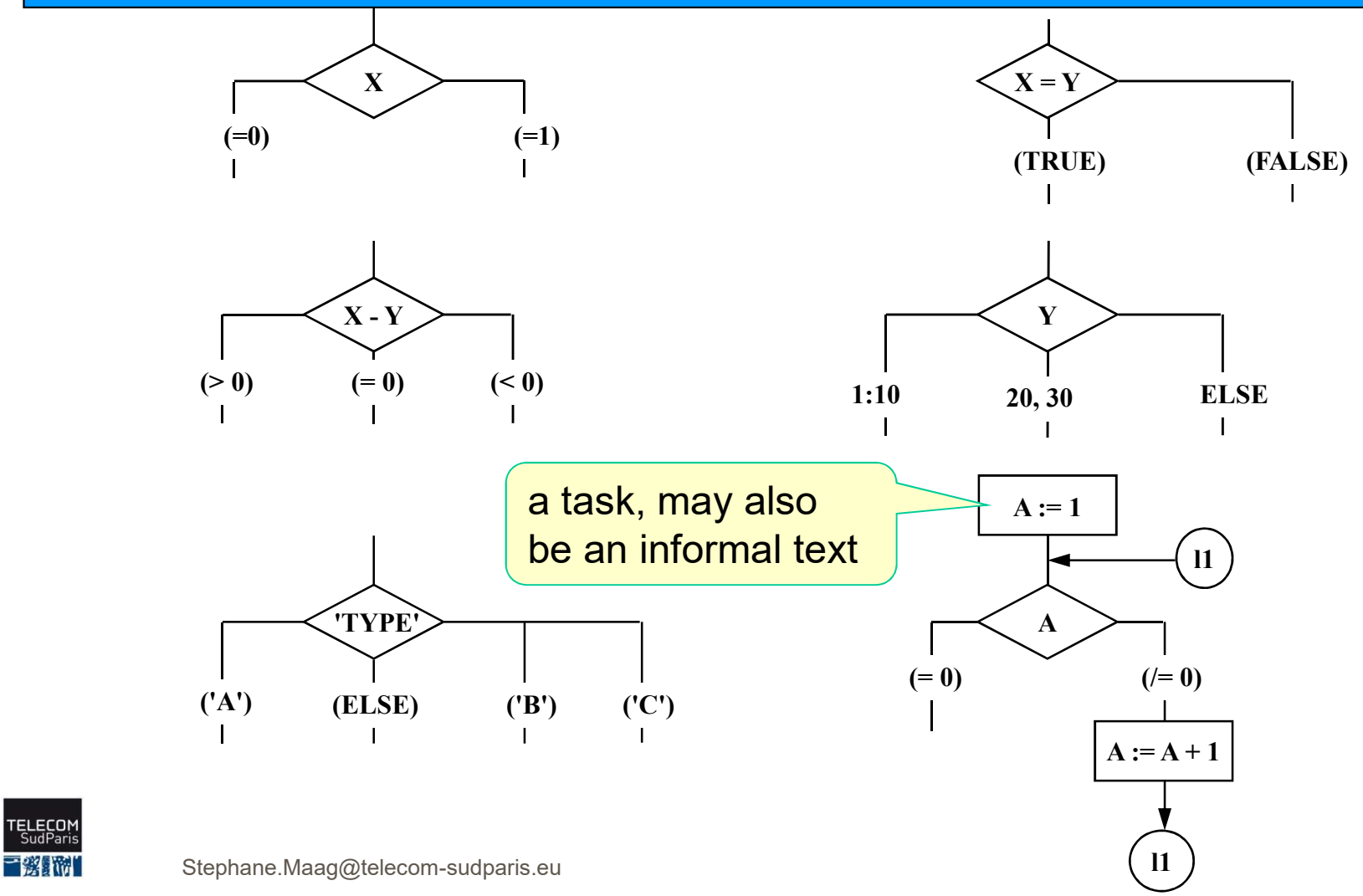
- signal can only be consumed if the condition is true, otherwise it is saved.
- ! The expression may not depend on current input signal parameters: only the *previous* value is accessible

Input - priority

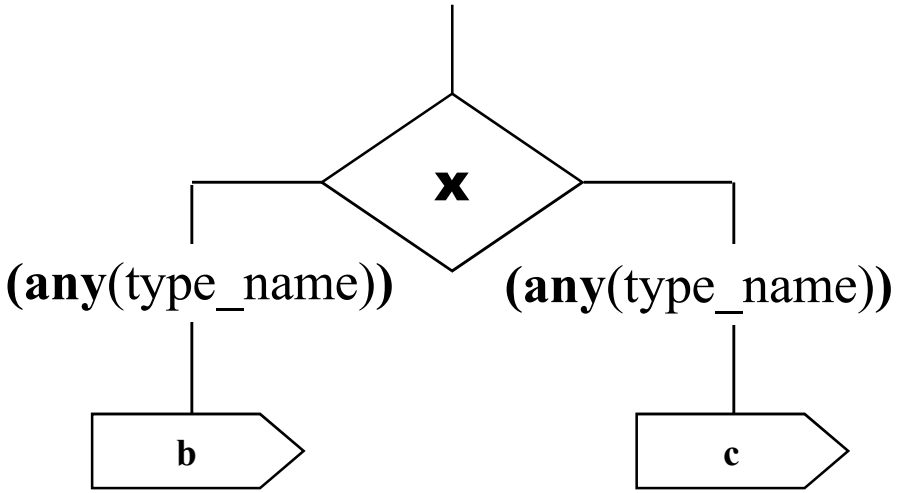


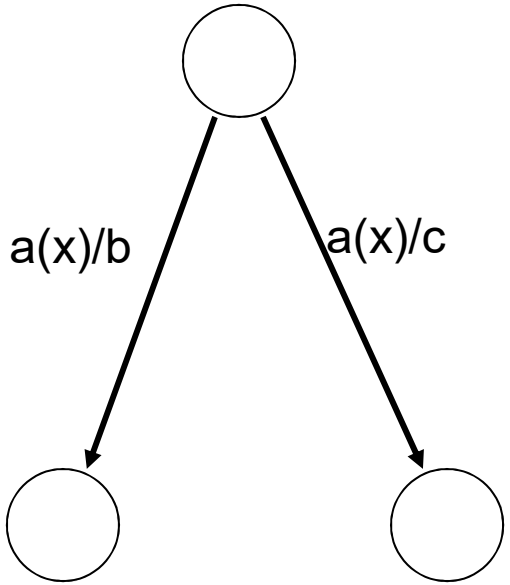
Priority signals are processed
prior to the other signals in the queue

Decisions


X
(=0)
(=1)
X = Y
(TRUE)
(FALSE)
X - Y
(> 0)
(= 0)
(< 0)
Y
1:10
20, 30
ELSE
a task, may also
be an informal text
A := 1
l1
'TYPE'
('A')
(ELSE)
('B')
('C')
A
(= 0)
(/= 0)
A := A + 1
l1

Non-deterministic transitions

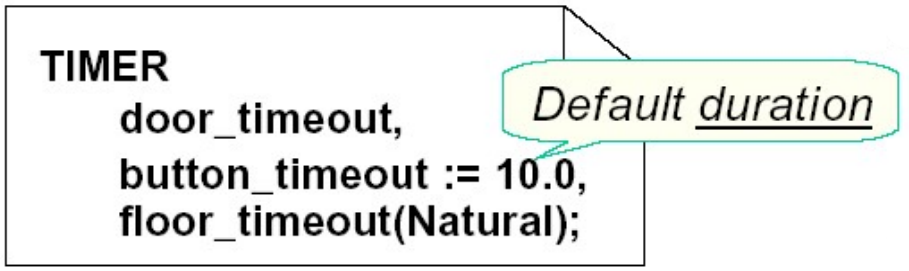




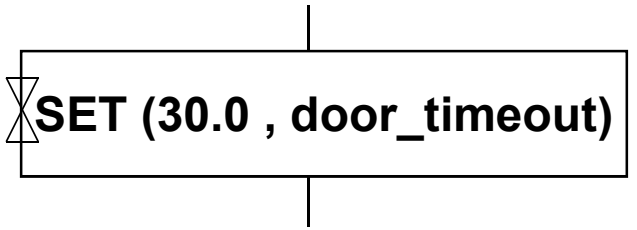
Non-deterministic transitions
are used to describe random events

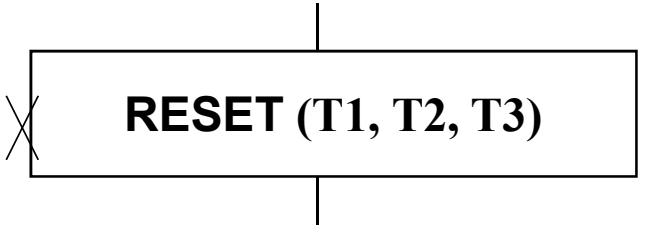
Express the Time in SDL

- A Timer is a meta-process able to transmit signals on demand to the process.

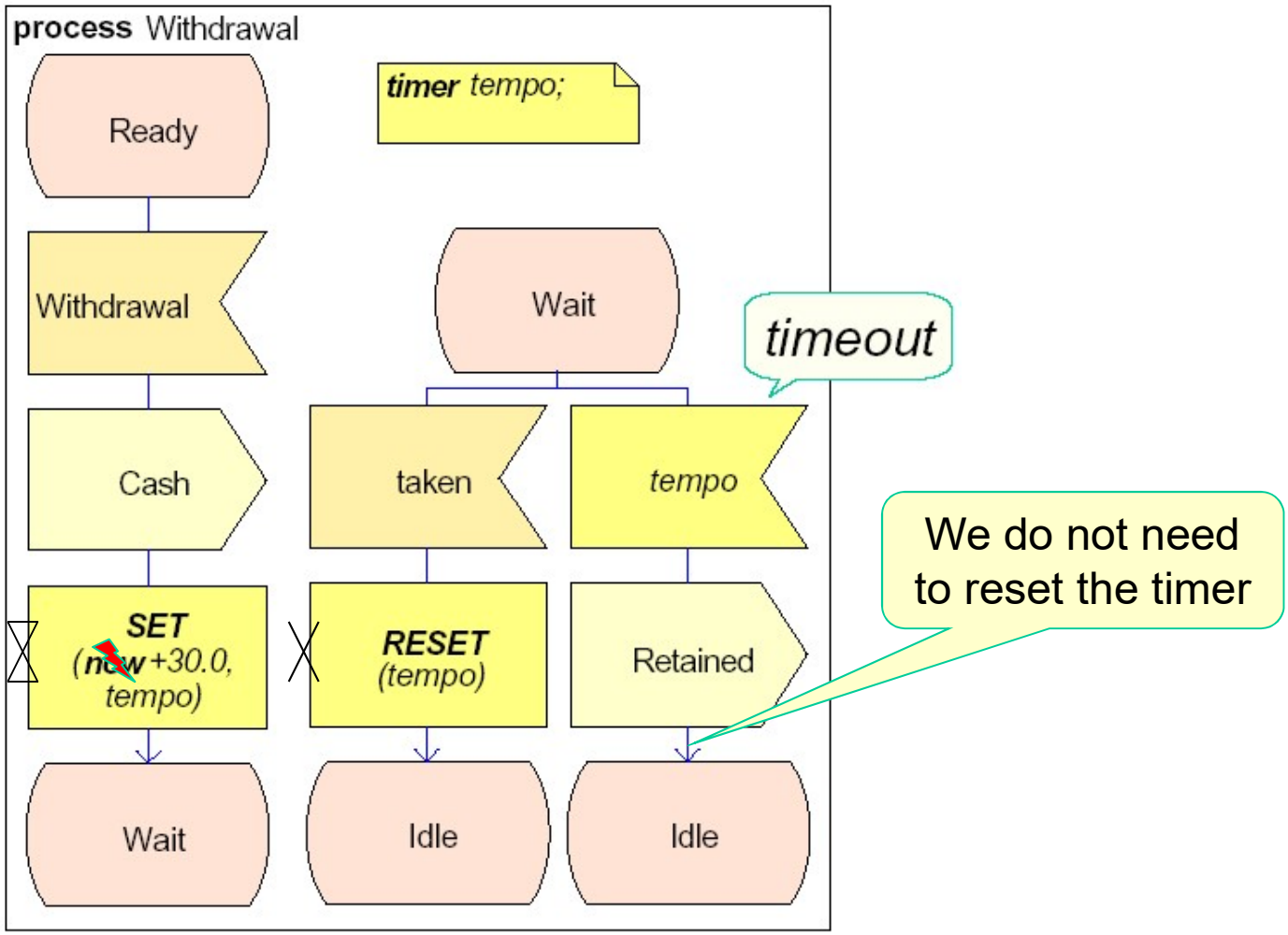


- The RESET also removes the corresponding signal from the process queue (case of an expired TIMER, but the signal is not consumed yet.



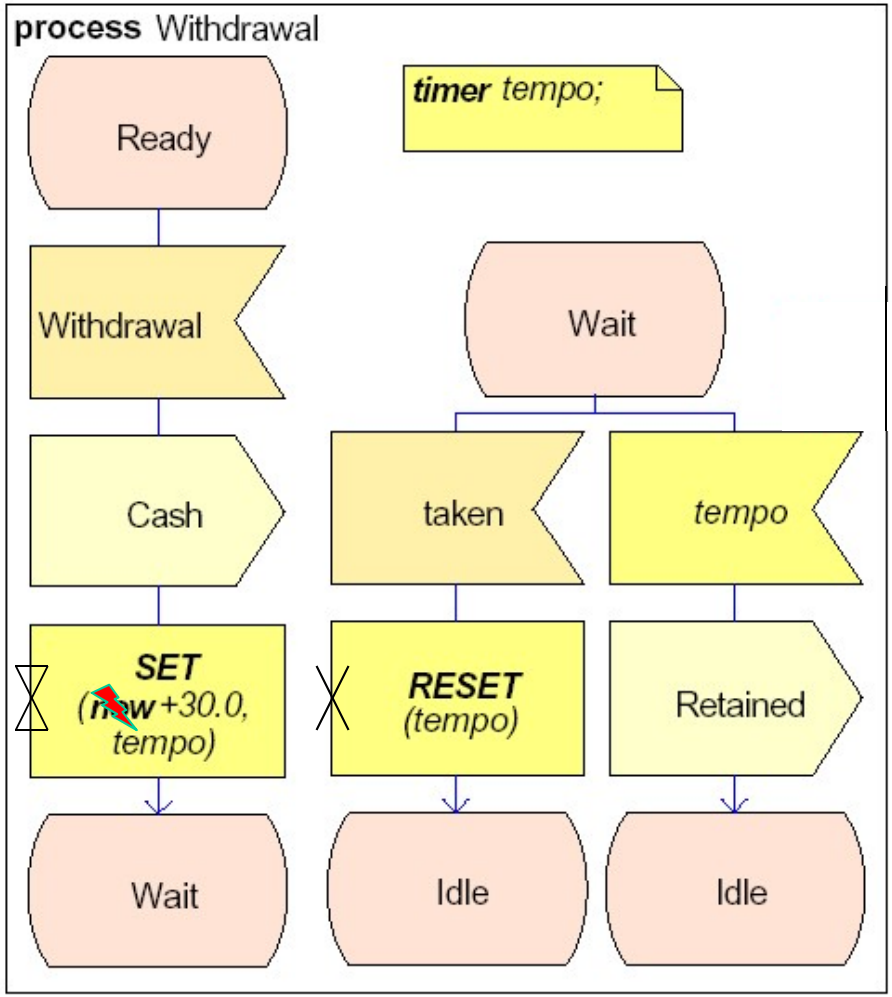


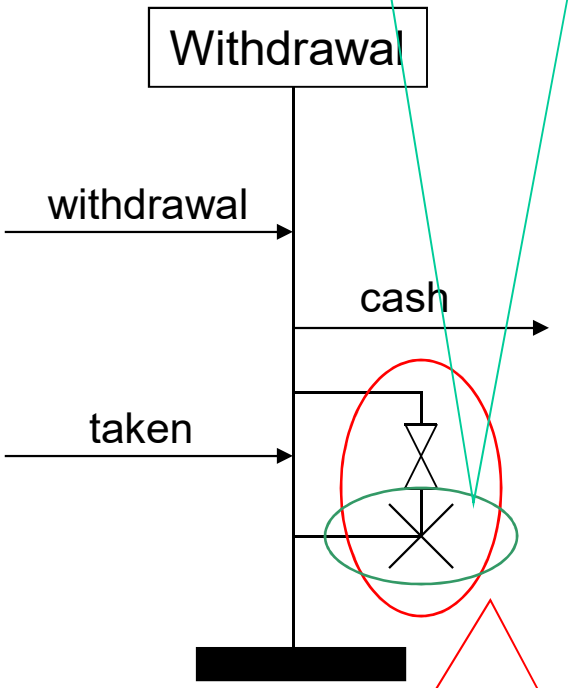
Use of Timers



Mapping with MSC

•Delete the process
•signals remaining in the queue are lost
•future messages to thies process are lost

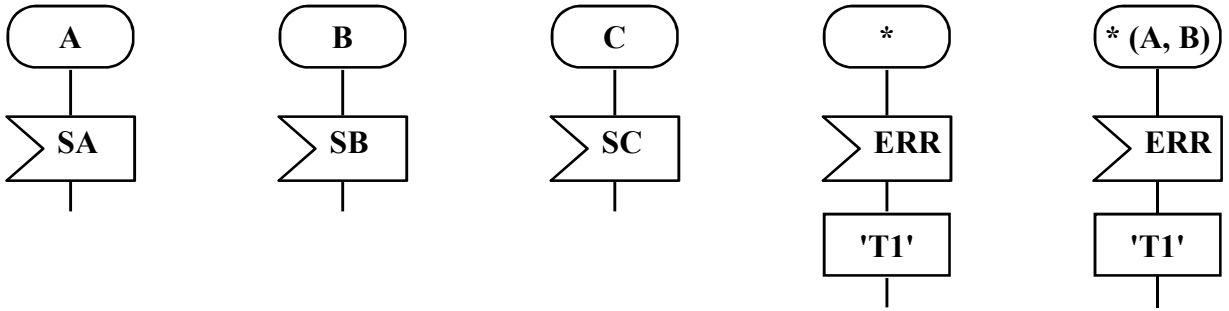




The timer is set and reset because the arriving of signal *taken*

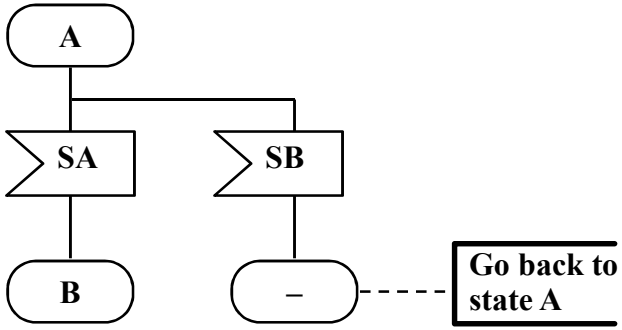
To ease the writing (1/2)

The transition associated to the state * is applicable with all the states, while the state *(A,B) is also applicable with all the states **except** A and B

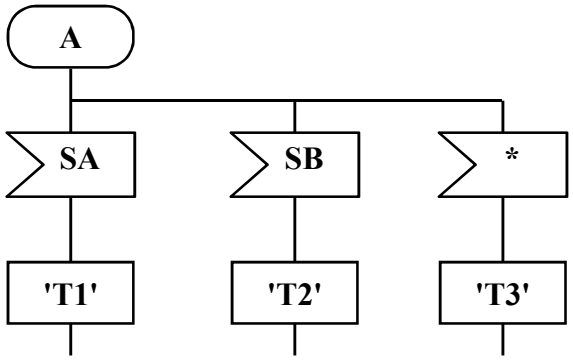


To ease the writing (2/2)

- To go back to the previous state



- Input *: represents all other signals



System simulation - Objectives

The model is now syntactically correct and semantically consistent. But it is good ?

From low costs to high quality:

- debugging
- evaluation of alternative solutions
- verification, detection of errors, comparison with MSC requirements.
- Test generation

$\Rightarrow$ to minimize the final costs

Two kind of simulation

Interactive

- step-by-step (debugging)
- access to all data
- MSC generation
- SDL tracking

Exhaustive

- fully automatic
- measures state and transitions coverage
- check properties
- reachability graph generation

•SDL, a FDT for complex system specification
•MSC to SDL
•SDL system
•SDL notations
•SDL process
•From the specification to the simulation
•**RTDS**

Pragmadev Studio

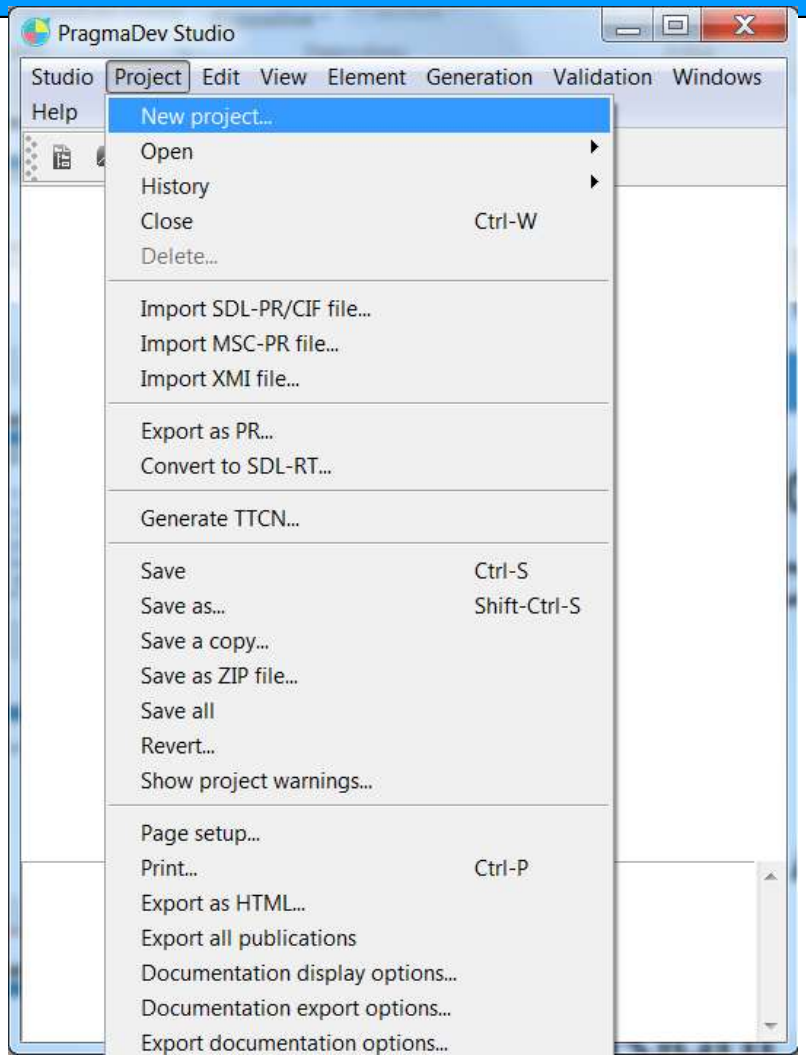
- A Pragmadev tool
- The tool allowing the edition from the requirements
- Architectural and behavioral design
- Model checking capabilities,
- Traceability information.
- Code generation
- Testing
- TTCN3

GUI - PragmaStudio

Graphical User Interface

Then:

- New project
- SDL Z100 project



Stephane.Maag@telecom-sudparis.eu

Conclusion

- SDL, a language to specify complex systems. User-friendly with its PR/GR
- Powerful to express important protocols
- Allows to simulate system behaviors

- In the following : *on the road of instantiating and testing … on the road ..*

SDL the following ... Part 2

TMSP

Stéphane Maag

TELECOM SudParis

Objectives

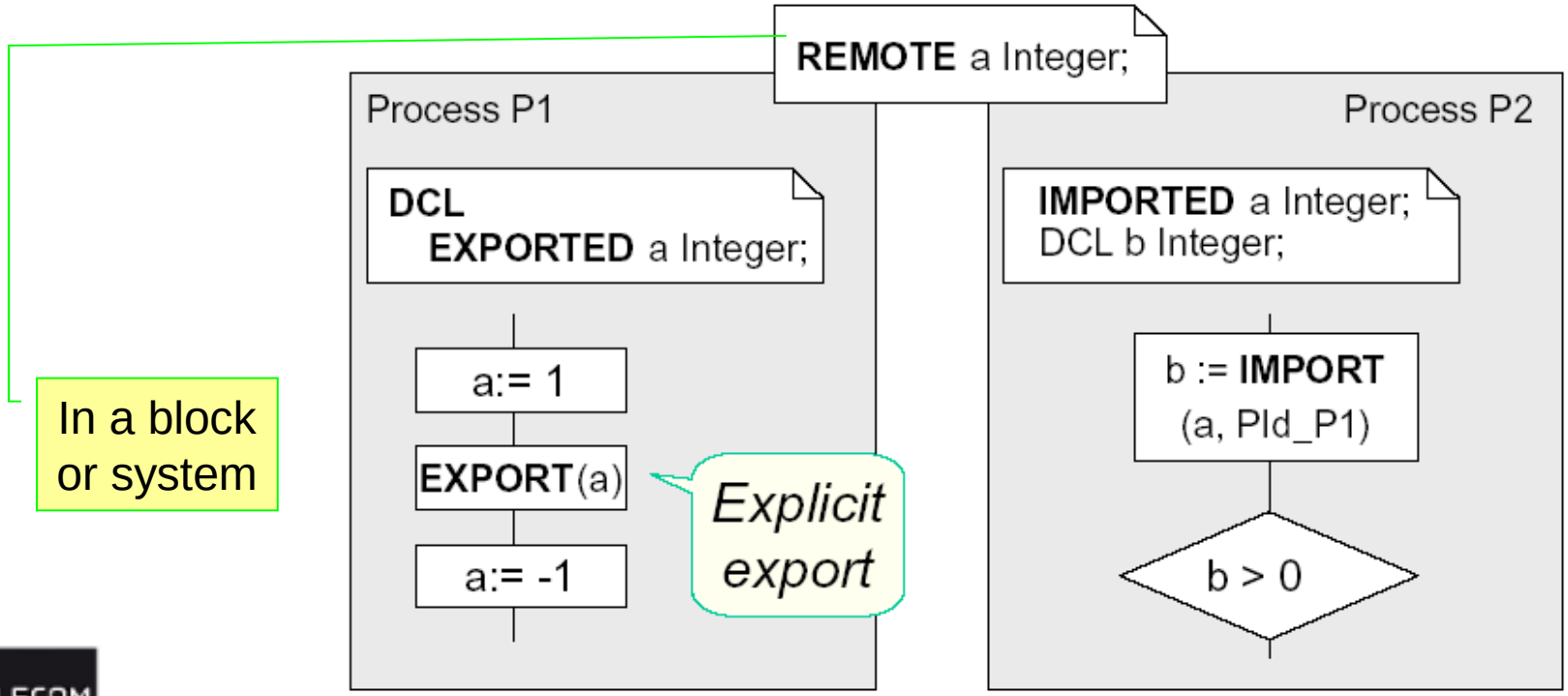
This course intends to make the participants discover:

- Structures
- Structural types
- Packages
- PID
- Procedures
- MacroDefinitions
- ASN.1 - Z.105

Stephane.Maag@telecom-sudparis.eu

Remote Variables

- **EXPORT-IMPORT**: to get the value of a variable of another process (implicit signals)



Stephane.Maag@telecom-sudparis.eu

Definition of Structure

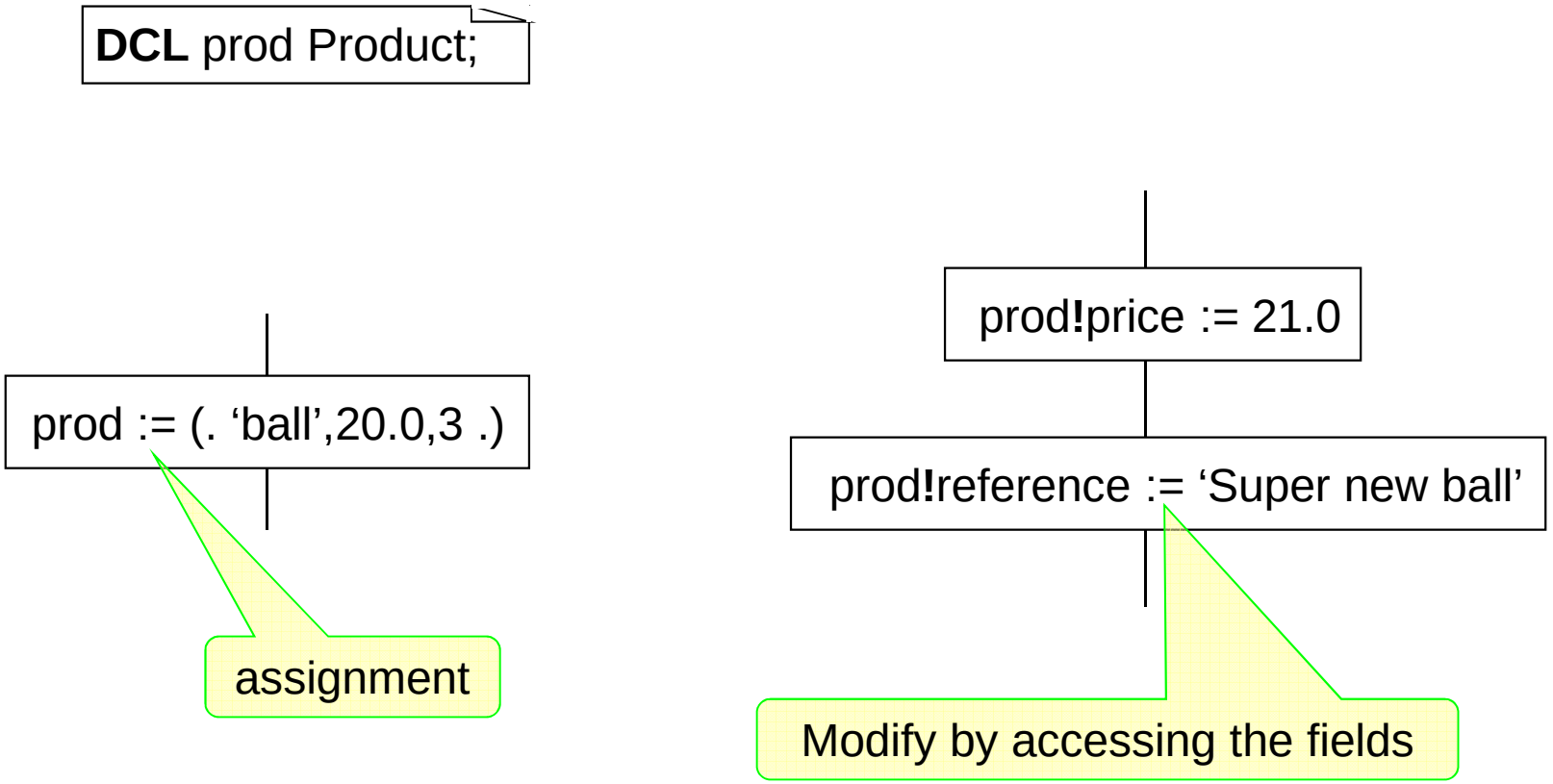
Structure with Fields

```
NEWTYPE Product
STRUCT
        reference CHARSTRING;
        price REAL;
        quantity INTEGER;
ENDNEWTYPE Product;
```

The field types
may be some structures

Stephane.Maag@telecom-sudparis.eu

Use of Structure



Stephane.Maag@telecom-sudparis.eu

Array Type

Type of
the index (**integer)**

```
NEWTYPE Product_T
ARRAY (Index_T,Product);
ENDNEWTYPE;
```

```
DCL prod_set Product_T;
DCL prod Product;
```

Initialization

```
prod_set := (. (. 'ball',20.0,3 .) .)
```

Modify

```
prod_set(1) := prod
```

Indexes are **integer**

Extract

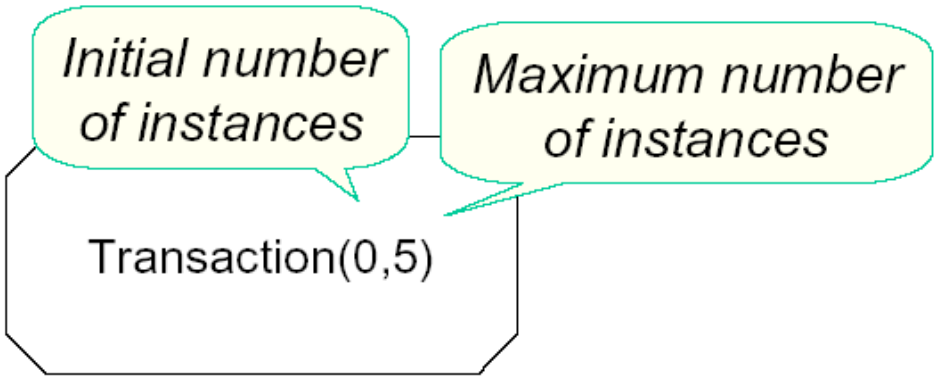
```
prod := prod_set(3)
```

TELECOM SudParis

Stephane.Maag@telecom-sudparis.eu

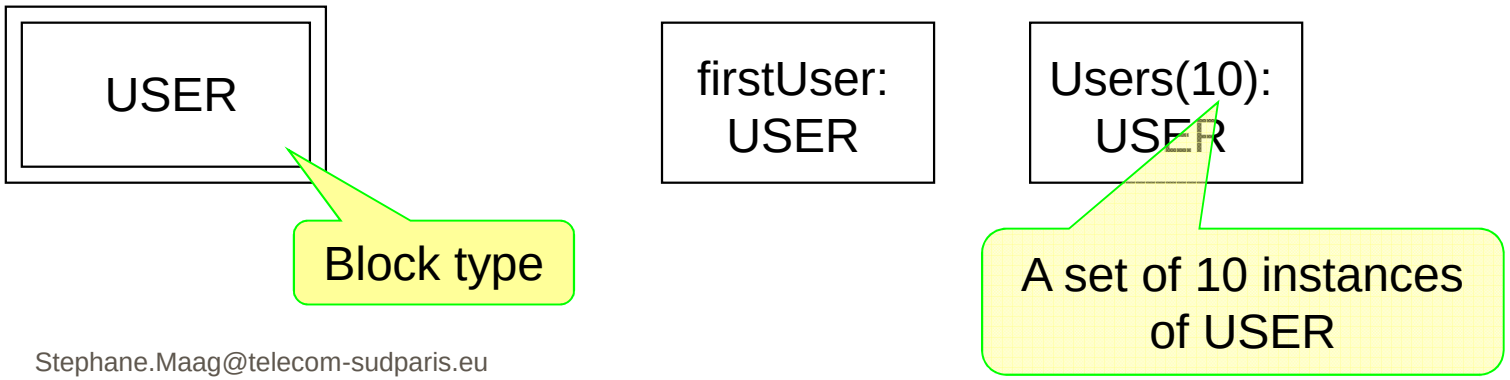
Process: Active Class

- SDL allows to generate process instances:
  - They are active objects
  - Perform their own actions
  - Manage their own data
- Several possible instances may run in parallel.
- To represent them:



Structural Types and Instances

- **Block type** or **Process type**
  - General description to reuse later
  - Allows to generate many block or process instances
  - Define the content of all instances
- We may define one instance or set of instances



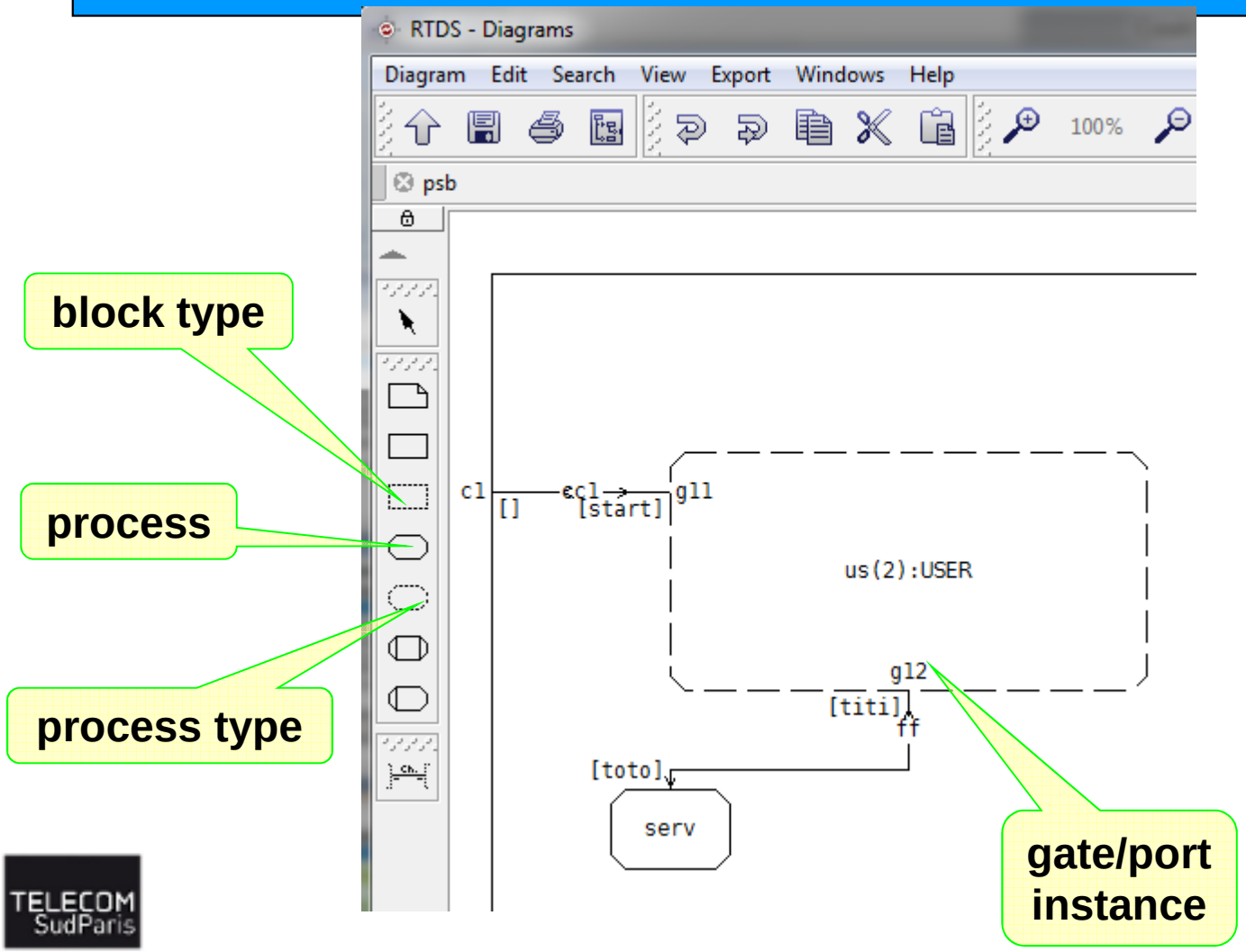
Stephane.Maag@telecom-sudparis.eu

Why Block and Process type ?

- They are defined once and used many times
  - a defined block may then be used in different systems
  - to define only once the content of several processes that run in different blocks
  - structural types and instances available for systems, blocks and processes

Stephane.Maag@telecom-sudparis.eu

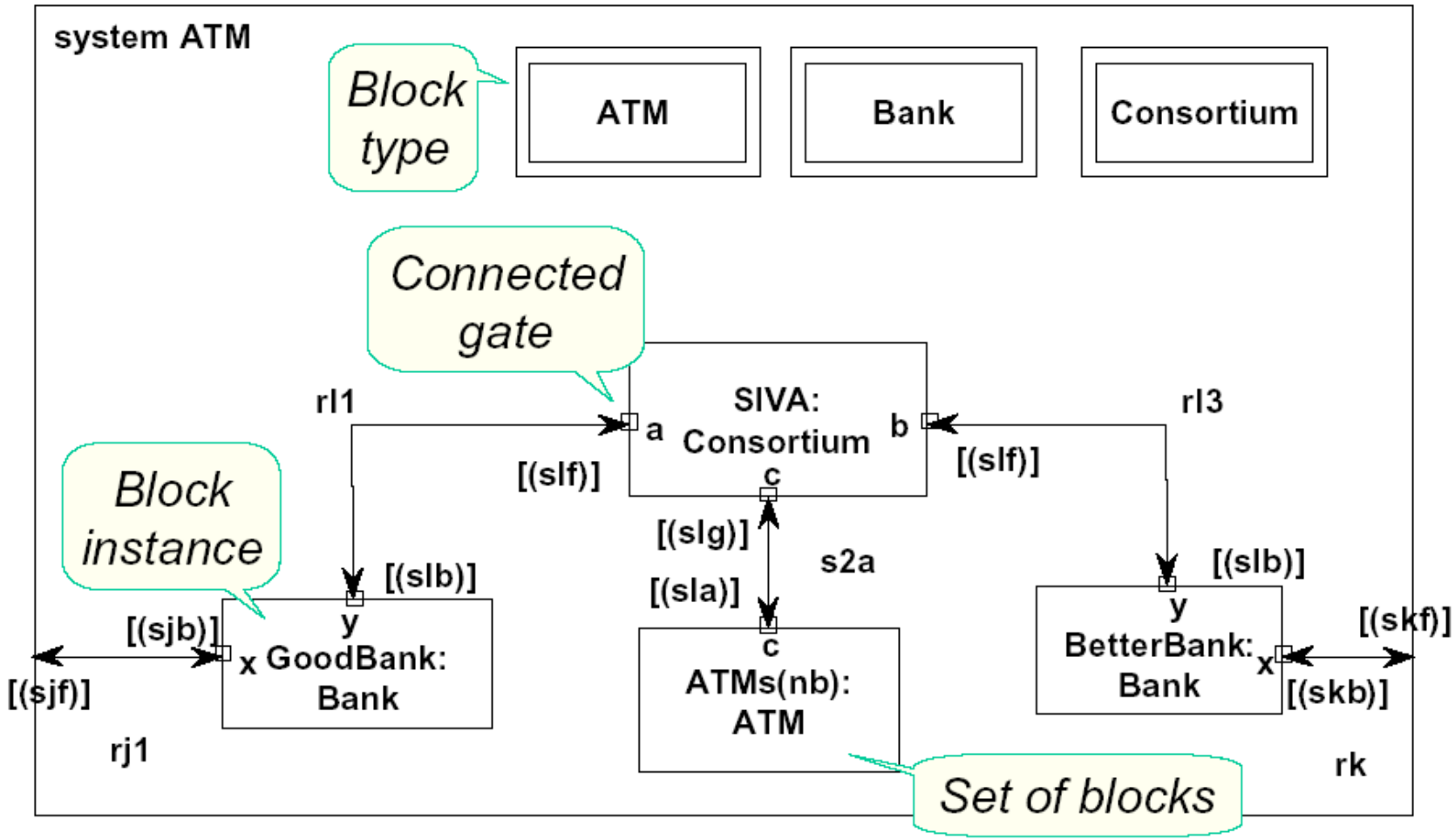
Structural Types in RTDS



- Structural types need gates.
- gates define, with channels, the signals sent and/or received.
- Instances are connected by channels through gates.

Stephane.Maag@telecom-sudparis.eu

Examples of instantiations



Stephane.Maag@telecom-sudparis.eu

Where to use structural types ?

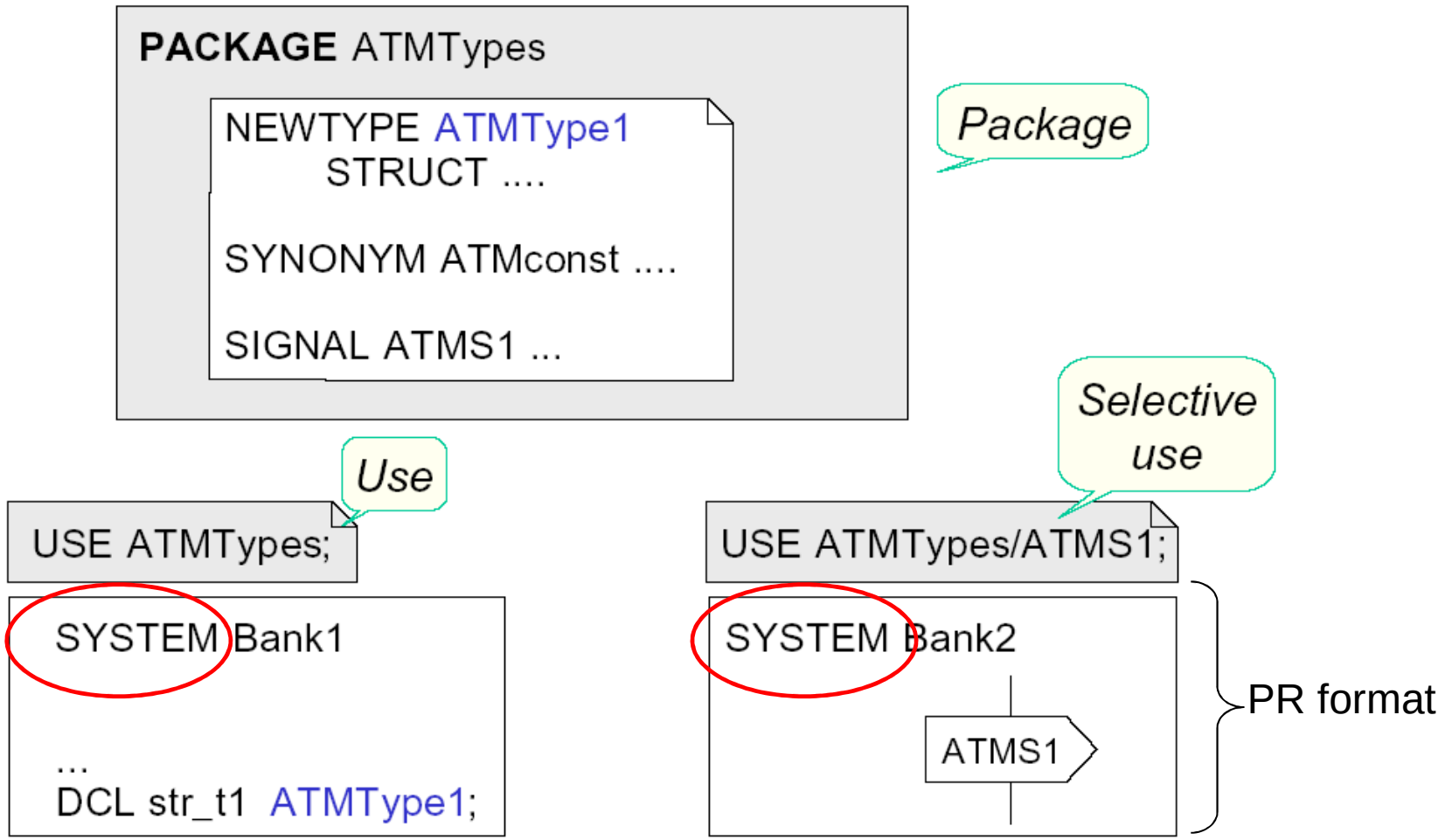
- In package: a set of types
  - Structural types (block types, process types,…)
  - Signals and lists
  - Constants
  - Data types

- A package allows to reuse types in several models.

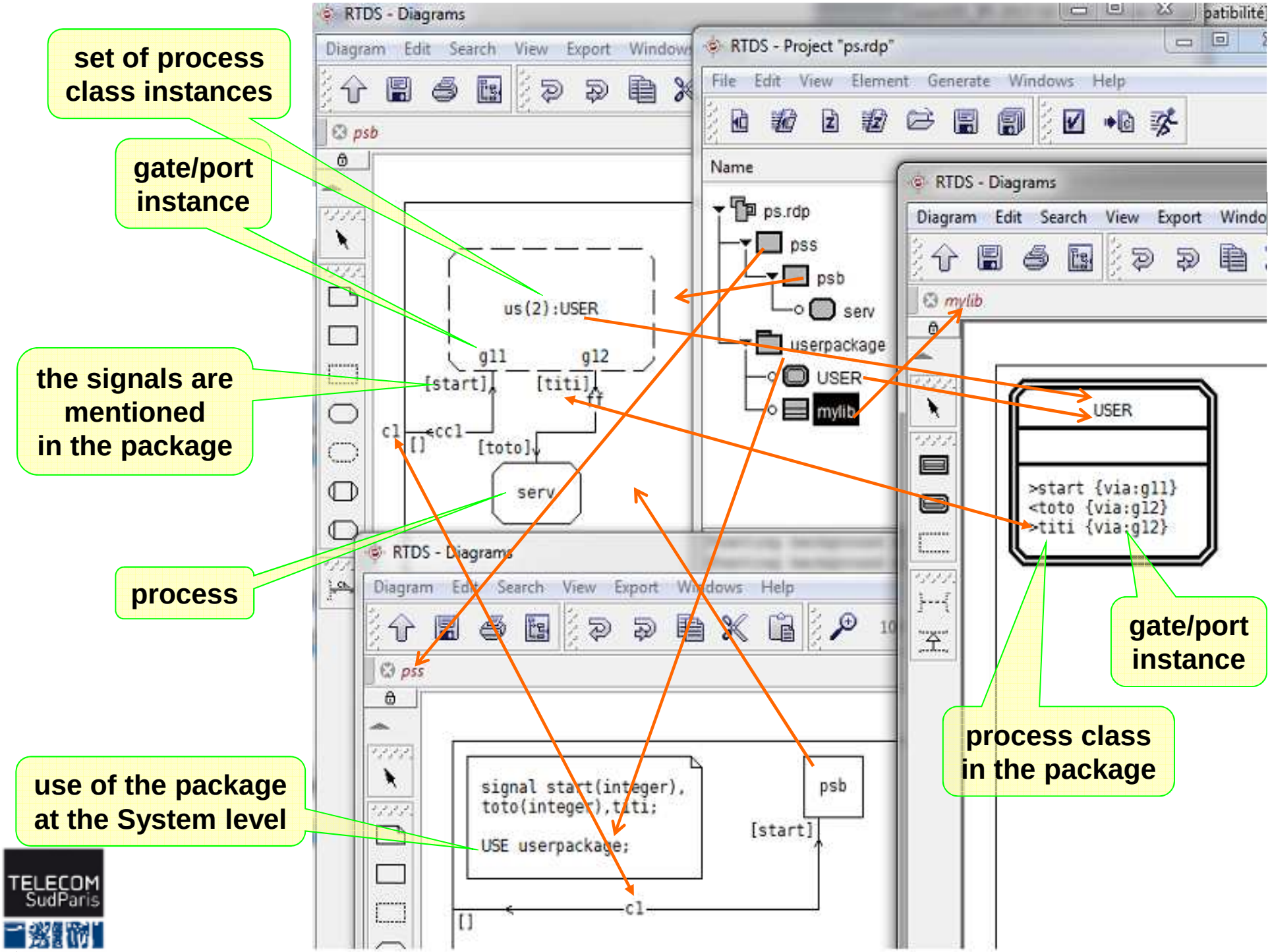
TELECOM SudParis

Stephane.Maag@telecom-sudparis.eu

Package example

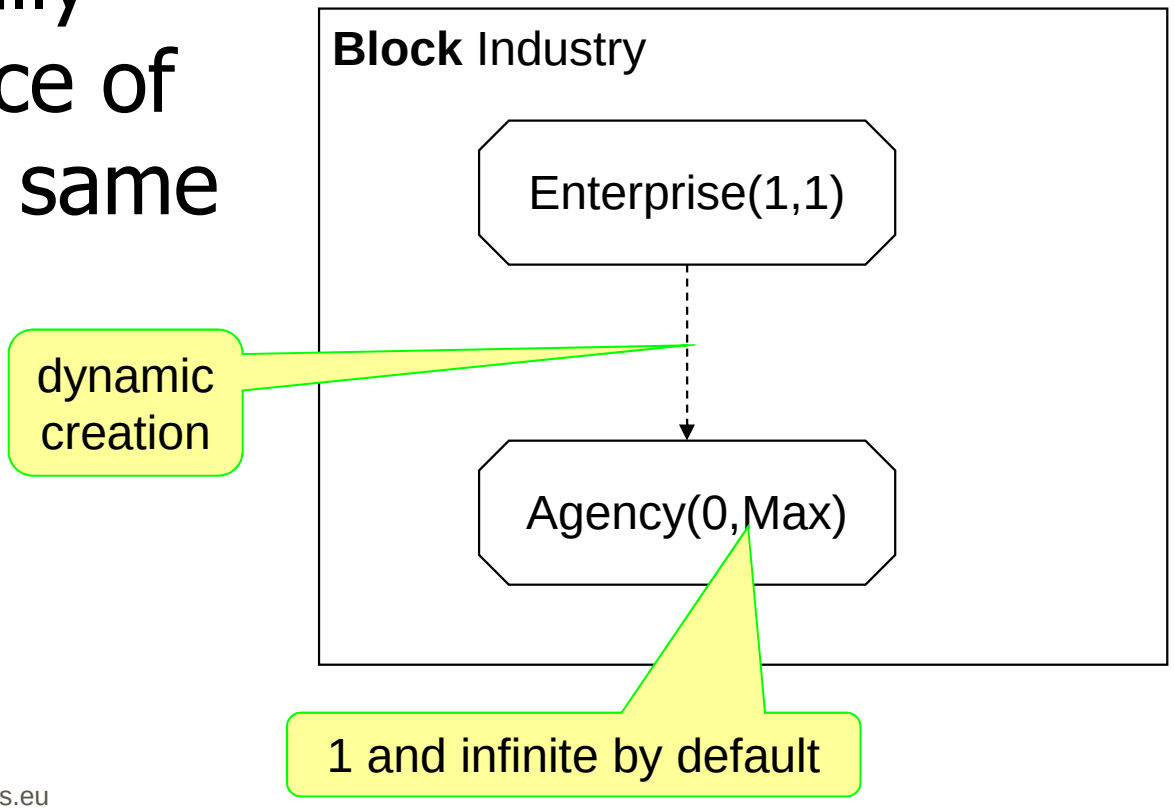


Stephane.Maag@telecom-sudparis.eu


set of process class instances
gate/port instance
the signals are mentioned in the package
process
use of the package at the System level
process class in the package
gate/port instance
RTDS - Diagrams
RTDS - Project "ps.rdp"
RTDS - Diagrams
RTDS - Diagrams
us(2):USER
g11
g12
[start]
[titi]
c1
cc1
[]
[toto]
serv
ps.rdp
pss
psb
serv
userpackage
USER
mylib
USER
>start {via:g11}
<toto {via:g12}
>titi {via:g12}
signal start(integer), toto(integer),titi;
USE userpackage;
psb
[start]
c1
[]
TELECOM SudParis

Dynamic Creation of Processes

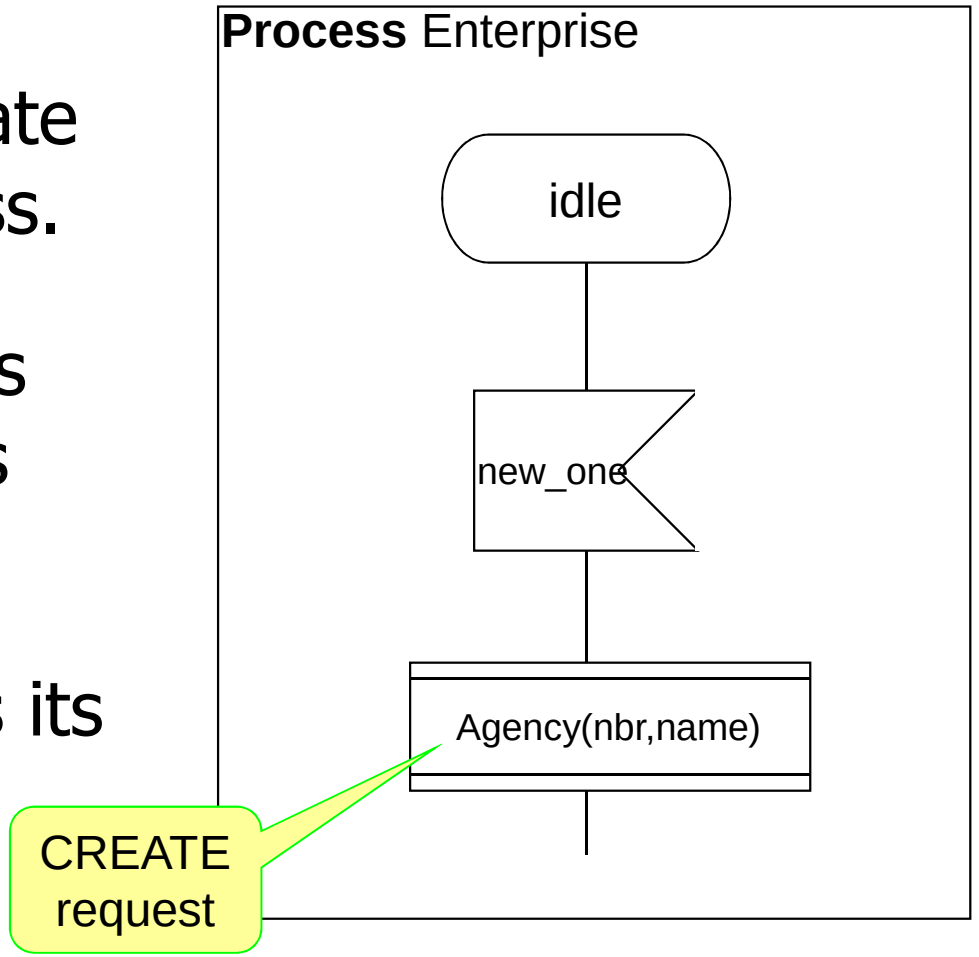
- Process instance may dynamically creates instance of process in the same block.



Stephane.Maag@telecom-sudparis.eu

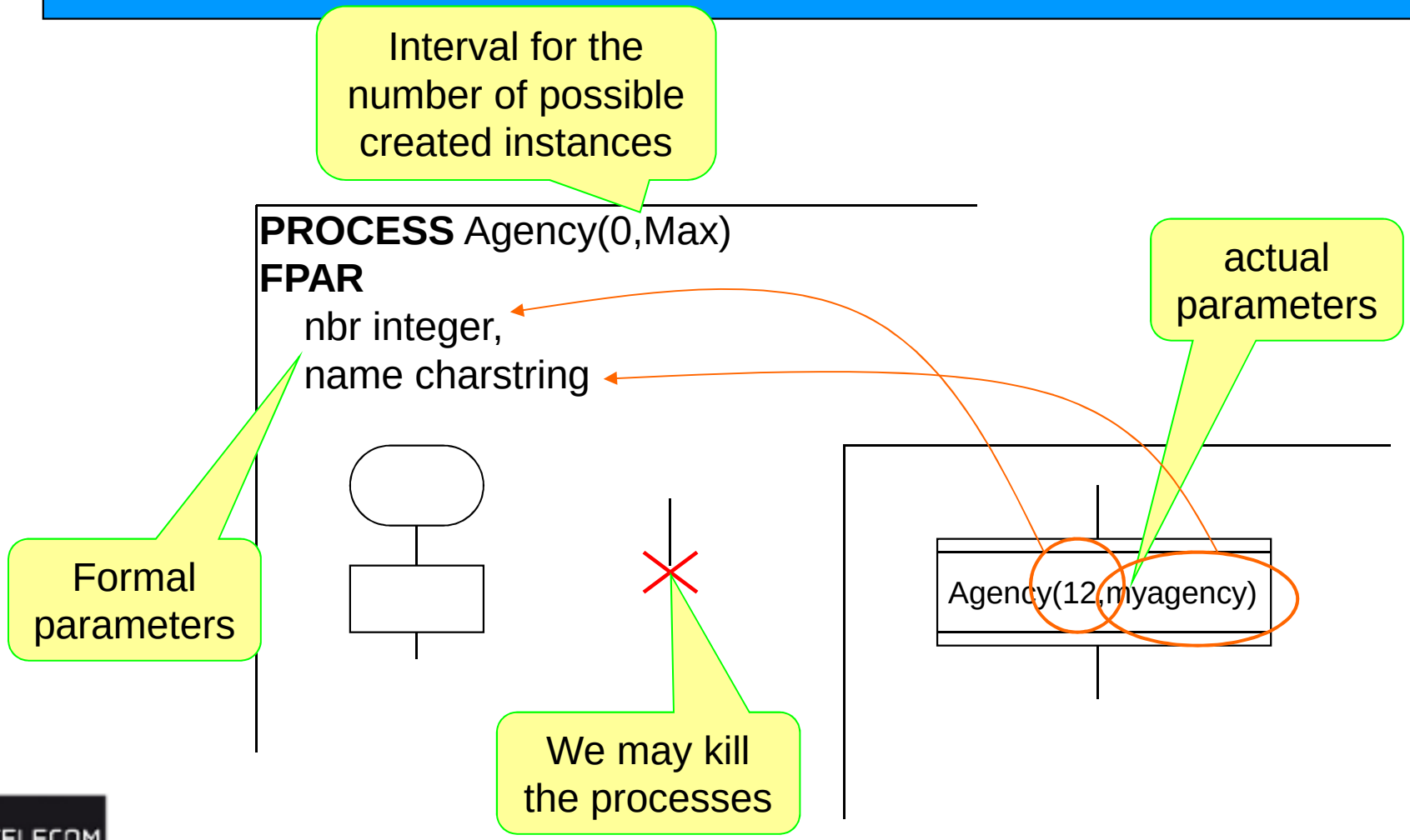
How to dynamically create processes

- The CREATE request provokes the immediate creation of the process.
- The created processes may carry parameters given by the creator.
- The new instance has its own new PID



Stephane.Maag@telecom-sudparis.eu

Process parameters

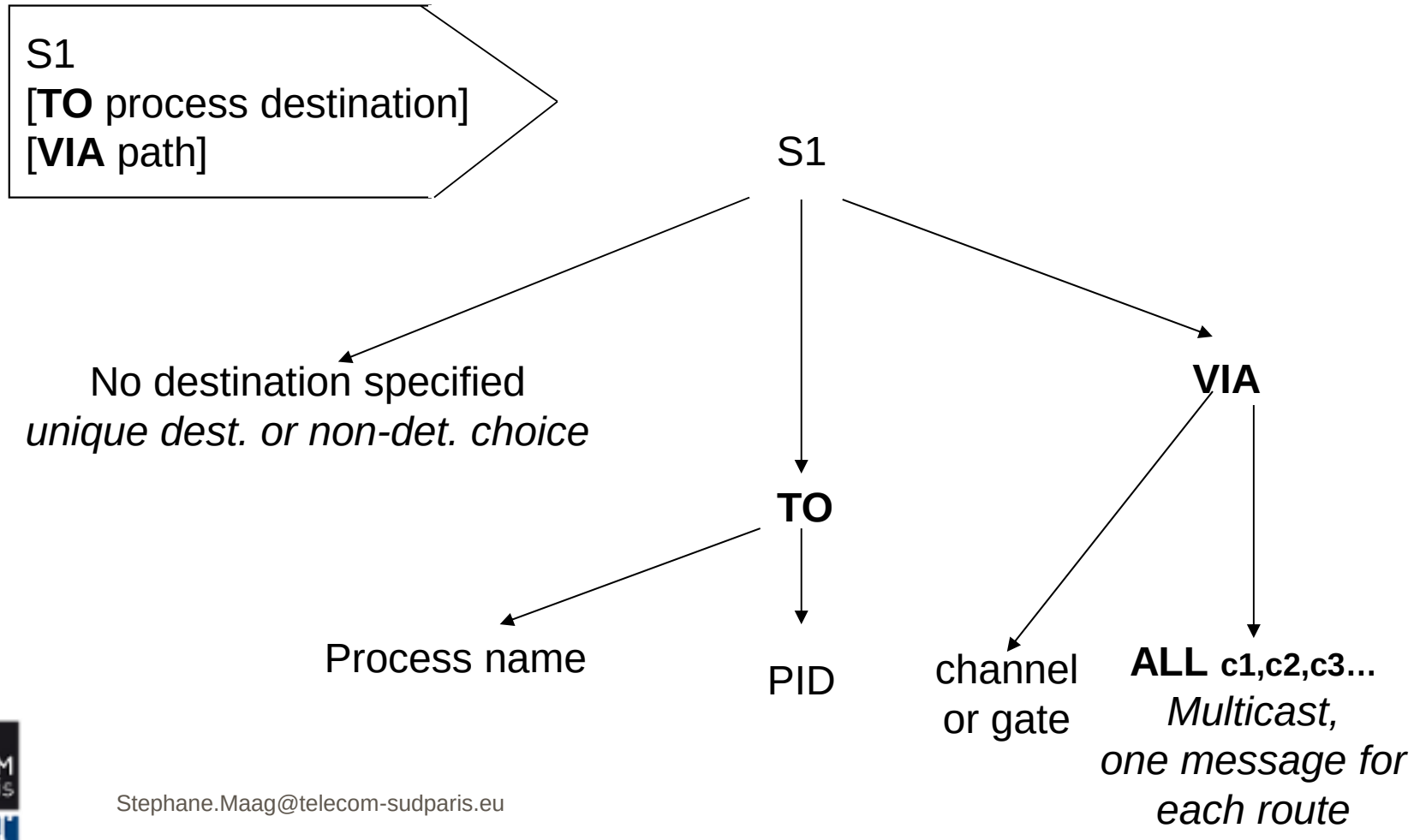


Stephane.Maag@telecom-sudparis.eu

Process IDentification

- The PID is the unique identifier of each instance of process.
  - Remember … PID is a predefined type !
  - The PID cannot be modified
  - The PID type has one predefined constant: NULL
- PIDs are used for communication in case of many possible receivers.
  - Client/server, mobile topologies (broadcast),…
  - signals that are both sent and received, …
  - …

Process destination for Output

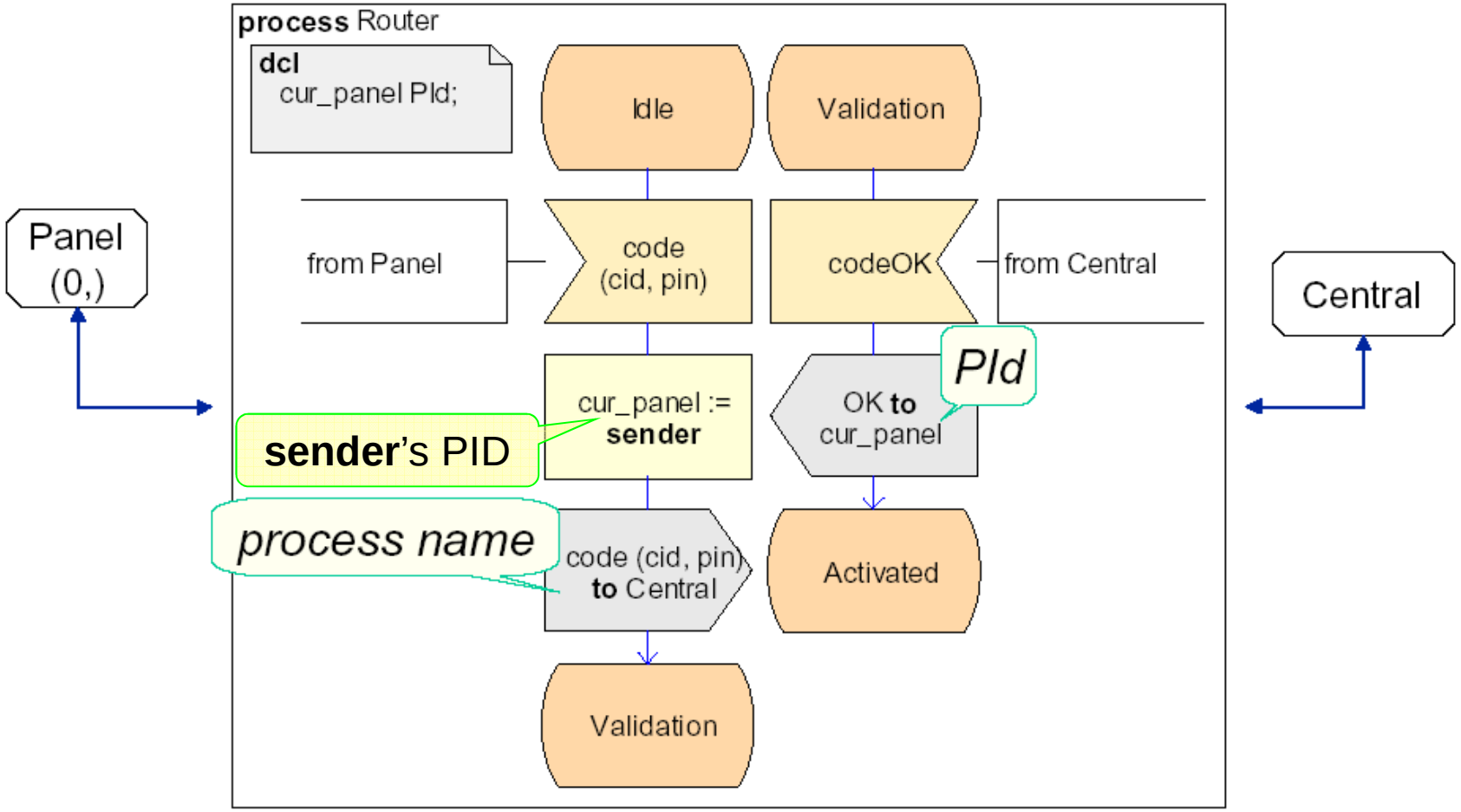


Stephane.Maag@telecom-sudparis.eu

Predefined PID expressions

- SELF: PID of the process itself
- OFFSPRING: last process instance created by itself. If none was created then OFFSPRING is NULL
- PARENT: PID of parent process. If SELF was not dynamically created, then PARENT is NULL
- SENDER: PID of the process that has sent the last consumed signal by SELF. If no consumed signal the SENDER is NULL

Example of PID use



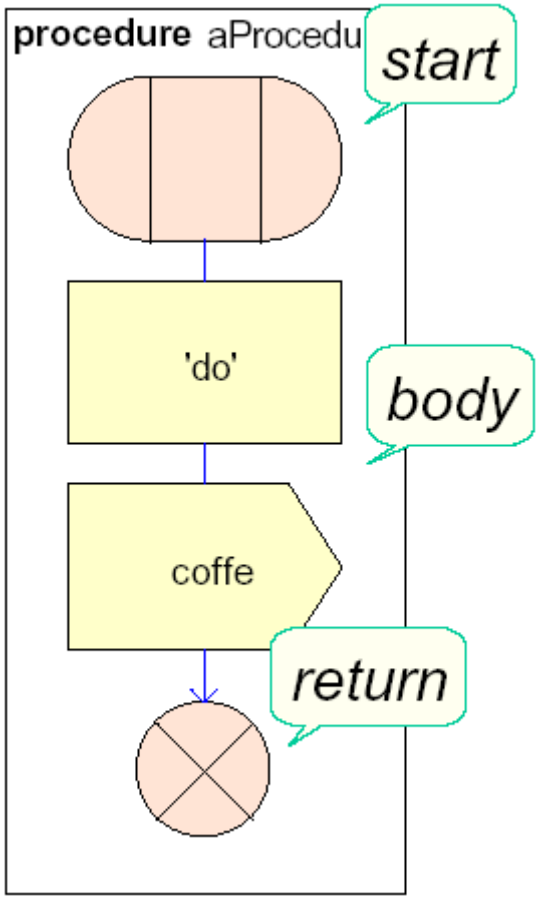
Stephane.Maag@telecom-sudparis.eu

Procedures

- Use to factorize and parameterize actions
- Encapsulation, abstraction
- Allow to reduce the EFSM size
- **Executed in their owning process**

Procedure definition

- It is described like process, using EFSM
- It is executed using the queue of its own process
- It does not have any PID (even the one of the process)
- Signals are sent to process only
- It may have **local variables**
- It can be **defined** at process, block or system level
- No Stop in a procedure

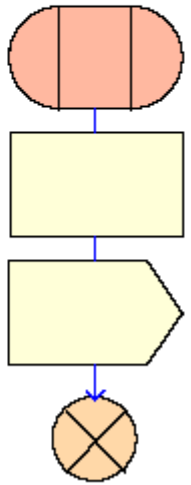


Stephane.Maag@telecom-sudparis.eu

Procedure parameters

- Introduced by **FPAR** and **IN** or **IN/OUT**:
  - **FPAR**: parameters of the procedure
  - **IN/OUT**:
    - by reference
    - it means that the parameter may be modified
  - **IN**:
    - by value
    - it means that the caller may not see the changes
  - by default the parameter are **IN**.

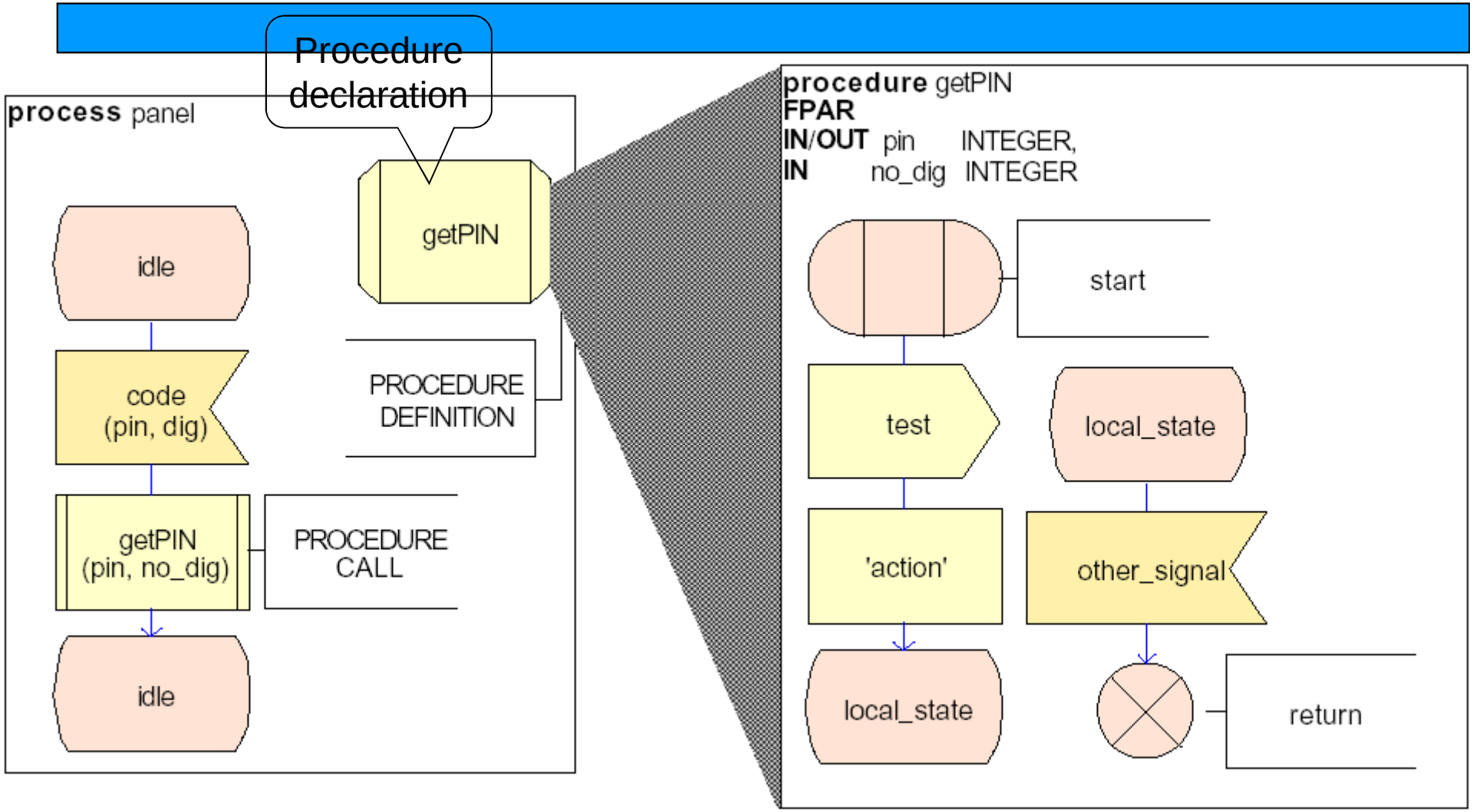
```
PROCEDURE GetPIN
FPAR
    IN/OUT pin integer,
    IN no_dig integer
```



TELECOM SudParis

Stephane.Maag@telecom-sudparis.eu

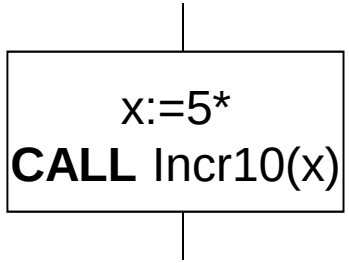
Procedure Example

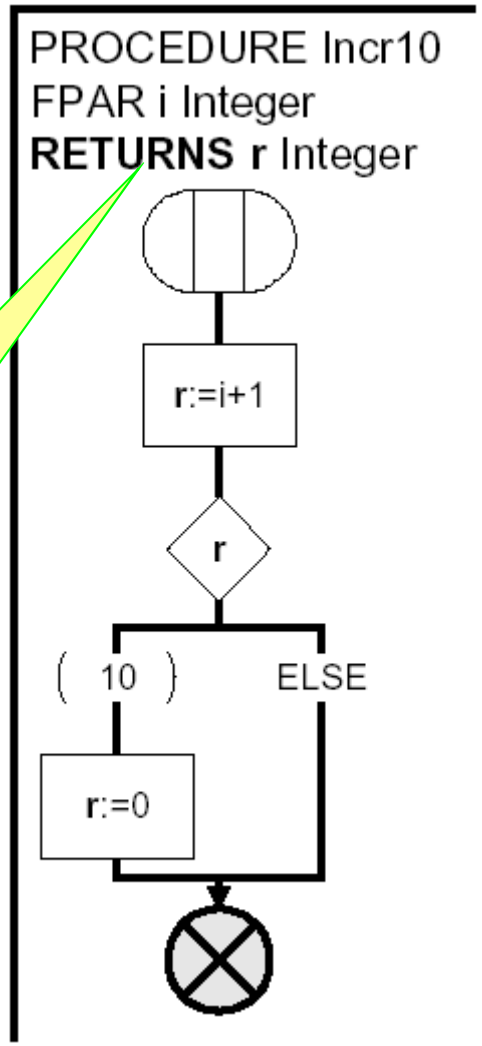


Stephane.Maag@telecom-sudparis.eu

Procedures as classic functions

- They may be called as classic functions in expressions: allows to return explicit values.





a value return

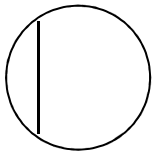
Stephane.Maag@telecom-sudparis.eu

MacroDefinition

- The Macros allow to treat the repetition of code, a description, a behavior that is often repeated,
- Used only within processes or procedures,
- May have formal parameter, it is necessary to transmit them.

Call of a macro

Macro inlet symbol

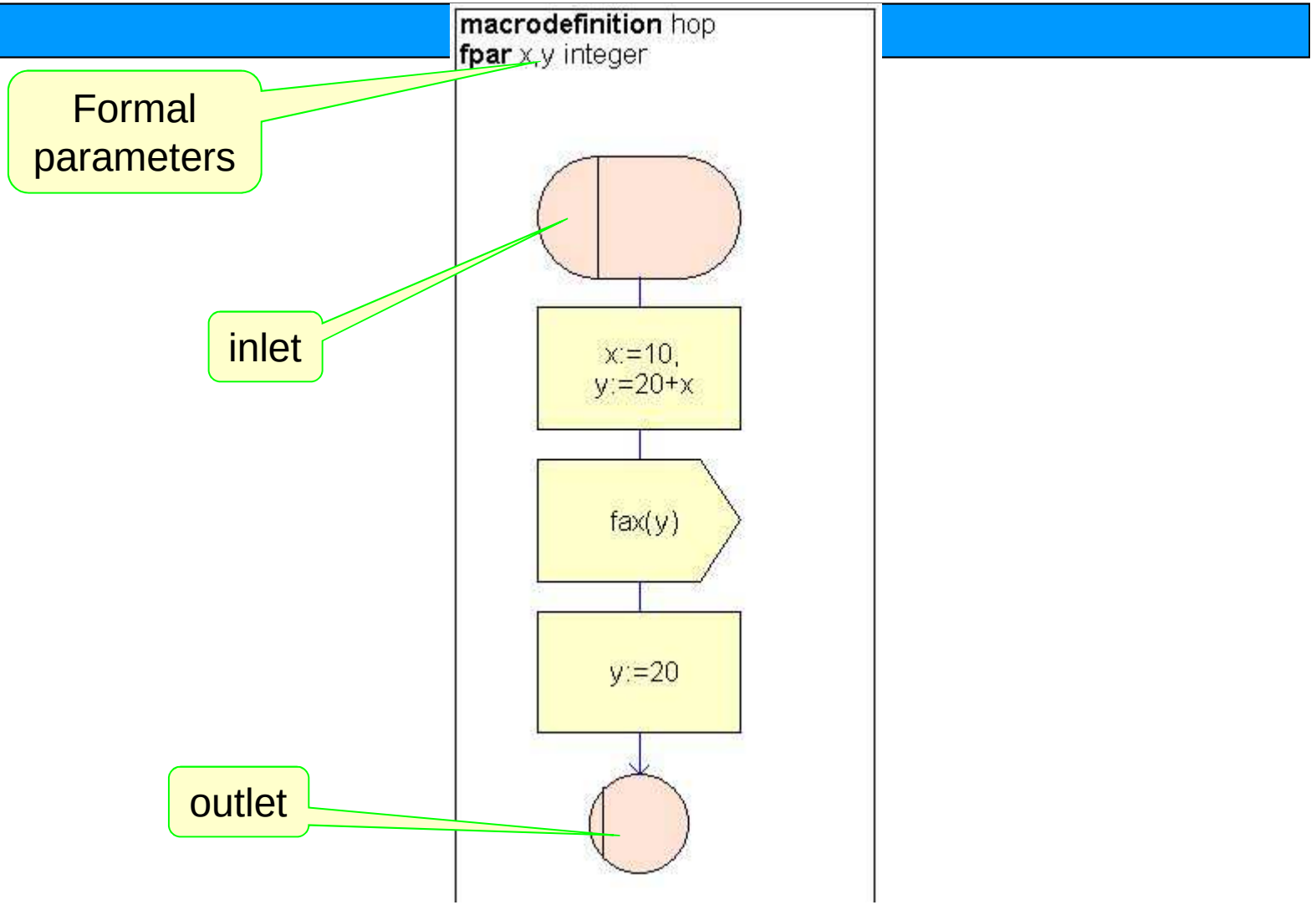


Macro outlet symbol

TELECOM SudParis

Stephane.Maag@telecom-sudparis.eu

Macrodefinition Example



Stephane.Maag@telecom-sudparis.eu

ASN.1 and SDL

- Z.105: Inclusion of ASN.1 in SDL
- Standard: ASN.1 is widely used in standards and can be part of the requirements
- Technically, ASN.1 allows to focus only on data: values, set of values, …

TELECOM SudParis

Stephane.Maag@telecom-sudparis.eu

Use of ASN.1 in Z.105

- ASN.1 declarations located in same box than SDL ones,
- ASN.1 and SDL declaration may be mixed,
- Z.105 is not case-sensitive,
- hyphens (“-”) cannot be used.

Stephane.Maag@telecom-sudparis.eu

ASN.1 predefined types

- INTEGER == INTEGER
- BOOLEAN == BOOLEAN
- REAL == REAL
- CHARSTRING == IA5String

SDL

ASN.1

NEWTYPE colors **LITERALS**
red, blue,black, yellow, white**;**
ENDNEWTYPE colors**;**

colors ::= **ENUMERATED**
{red, blue,black, yellow, white};

SYNONYM clearcolor colors
= white;

clearcolor colors ::= white;

Stephane.Maag@telecom-sudparis.eu

Composite types in ASN.1: Sequence types (Structure in SDL)

```
NEWTYPE T_Seq
STRUCT
        a BOOLEAN;
        b CHARSTRING OPTIONAL;
        c INTEGER DEFAULT 5;
ENDNEWTYPE T_Seq;
```

```
T_Seq ::= SEQUENCE {
        a BOOLEAN,
        b IA5String OPTIONAL,
        c INTEGER DEFAULT 5};
```

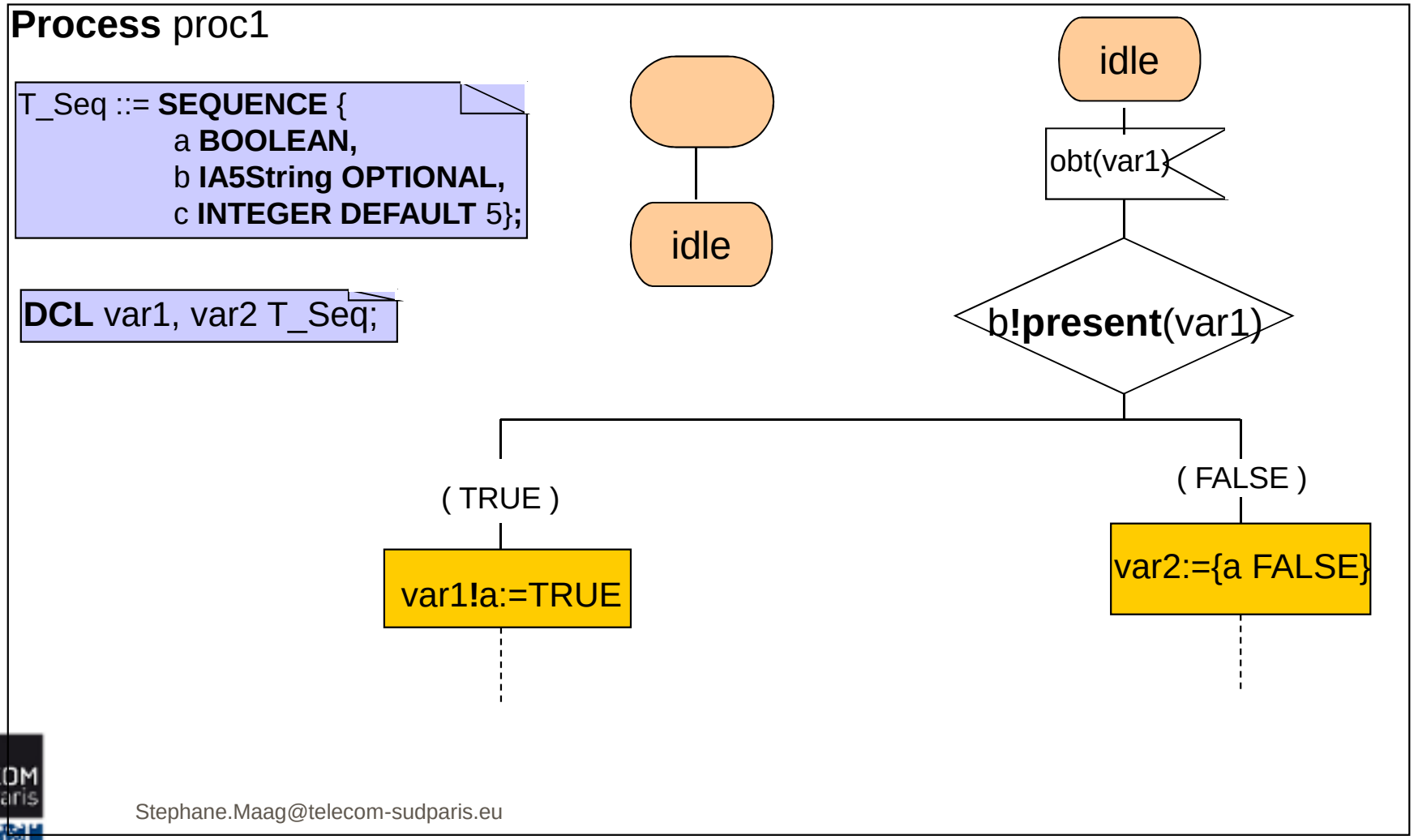
assignment

```
s1 T_Seq ::= {  a TRUE,
                b 'OK',
                c 12};
s2 T_Seq ::= { a FALSE,
                c 23};
s3 T_Seq ::= {a TRUE};
```

TELECOM SudParis

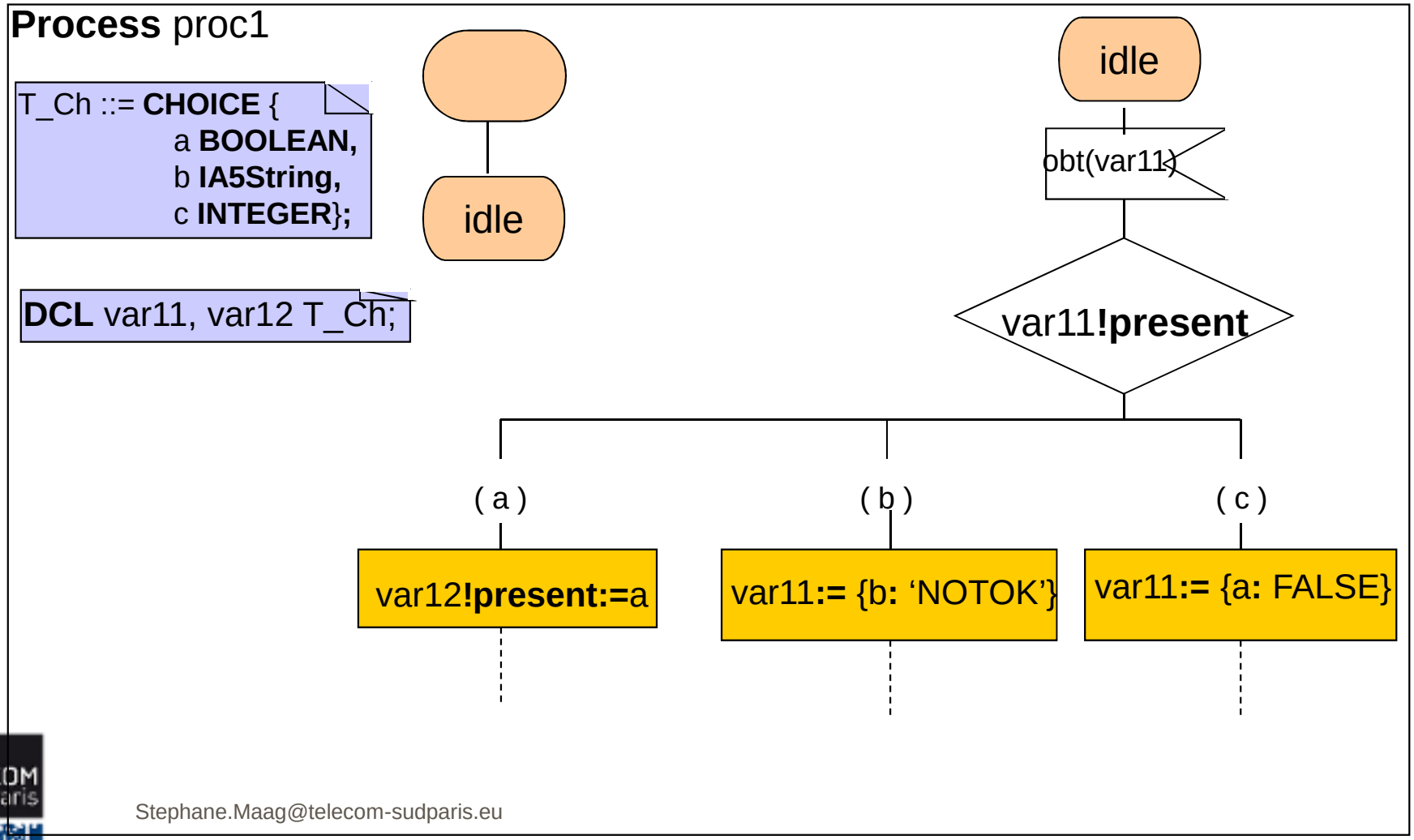
Stephane.Maag@telecom-sudparis.eu

Sequence Example



Stephane.Maag@telecom-sudparis.eu

Composite types in ASN.1: CHOICE types



Stephane.Maag@telecom-sudparis.eu

CONCLUSION

- Think to use structural types for reusability
- Process ID
- Readability with procedure and macrodefinitions
- CHOICE in ASN.1

Stephane.Maag@telecom-sudparis.eu

Exercises

Specify a process that receives a message *ATM_req* containing a data *atm_req* as a structure "*(quantity, ticket)"* where *quantity* is an integer and *ticket* is an optional character.

This process sends the output *OK* if *ticket* is received or *NOK* if not.

Stephane.Maag@telecom-sudparis.eu