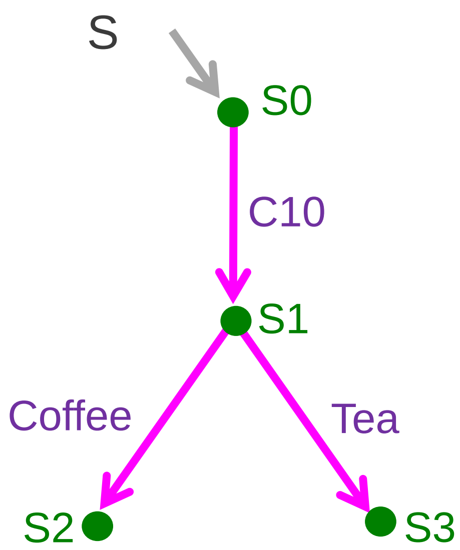
TXS

The Modelling Language for TorXakis

Hello World

```
CHANDEF  Chans
 ::=
       Input  :: String ;
       Output :: String
ENDDEF

PROCDEF  helloName [Inp, Outp :: String] ()
 ::=
           Inp ? name [[ strinre(name, REGEX('[A-Z][a-z]+')) ]]
      >->  Outp ! "Hello " ++ name ++ " !"
      >->  helloName [Inp, Outp] ()
ENDDEF

MODELDEF  Hello
 ::=
      CHAN IN   Input
      CHAN OUT  Output

      BEHAVIOUR
                 helloName [Input, Output] ()
ENDDEF

CNECTDEF  Sut
 ::=
      CLIENTSOCK

      CHAN OUT  Input   HOST "localhost" PORT 7890
      ENCODE    Input   ? s  ->  ! s

      CHAN IN   Output  HOST "localhost" PORT 7891
      DECODE    Output  ! s  <-  ? s
ENDDEF
```

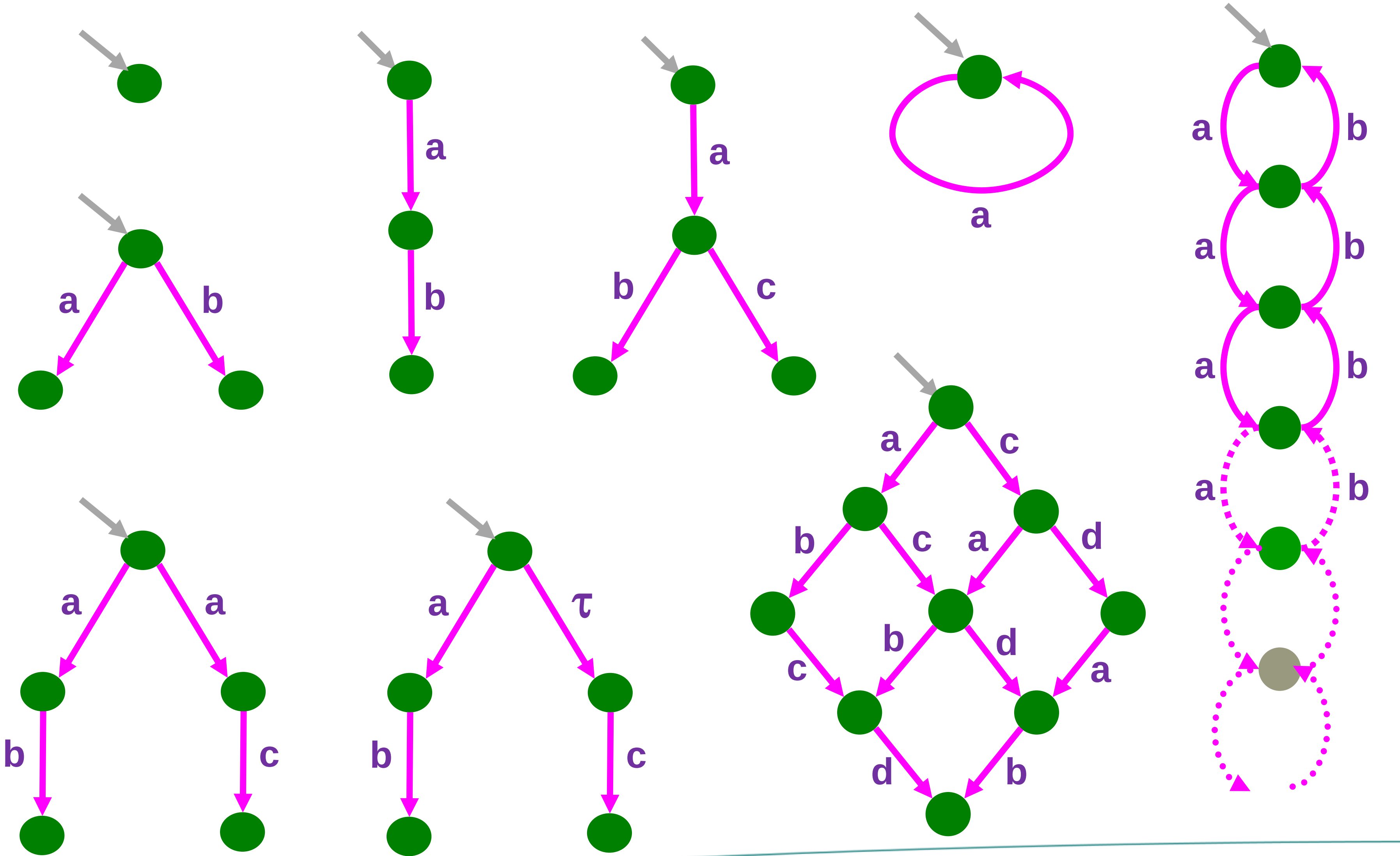
Representing LTS



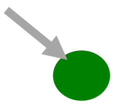
- Explicit : ⟨ { S0, S1, S2, S3 } ,
  {C10,Coffee,Tea} ,
  { (S0, C10, S1),
  (S1, Coffee, S2),
  (S1, Tea, S3) } ,
  S0 ⟩
- Transition tree / graph
- Language :

S ::= C10 >-> (Coffee ## Tea)

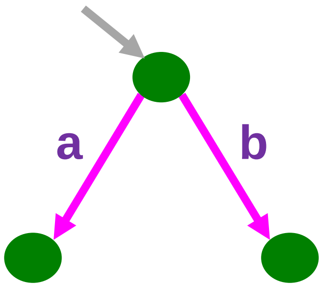
Labelled Transition Systems



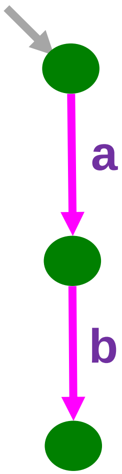
Representing LTS



STOP

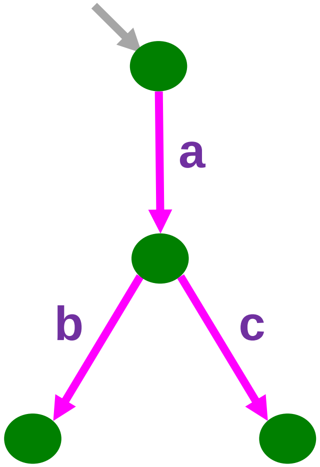


a ## b

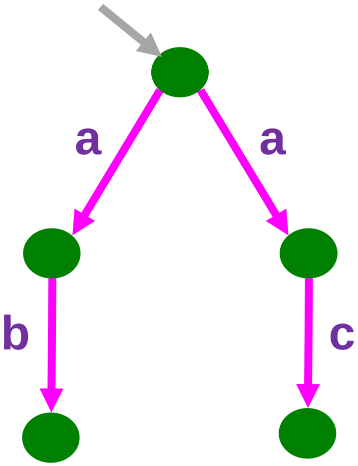


a >-> b

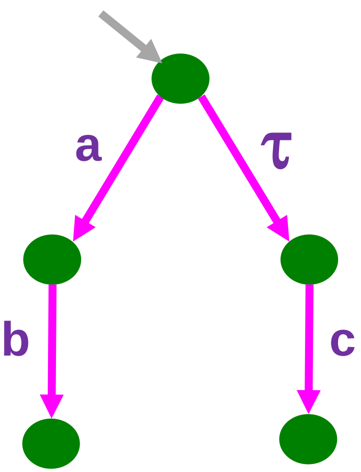
Representing LTS



a >-> (b ## c)



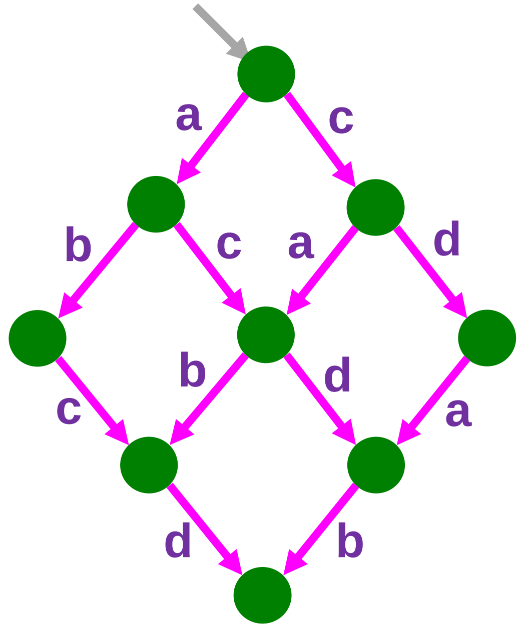
a >-> b ## a >-> c



a >-> b ## ISTEP >-> c

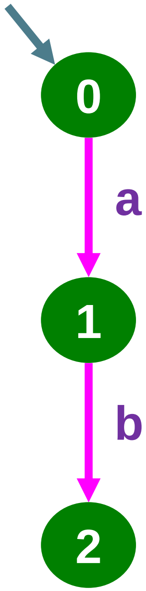
Representing LTS

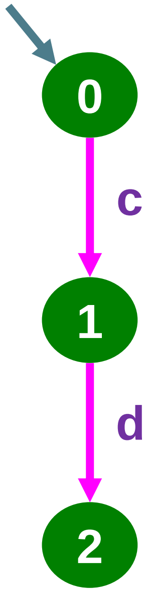
```
  a >-> ( b >-> c >-> d  ## c >-> ( b >-> d ## d >-> b ) )
##
  c >-> ( d >-> a >-> b  ## a >-> ( b >-> d ## d >-> b ) )
```

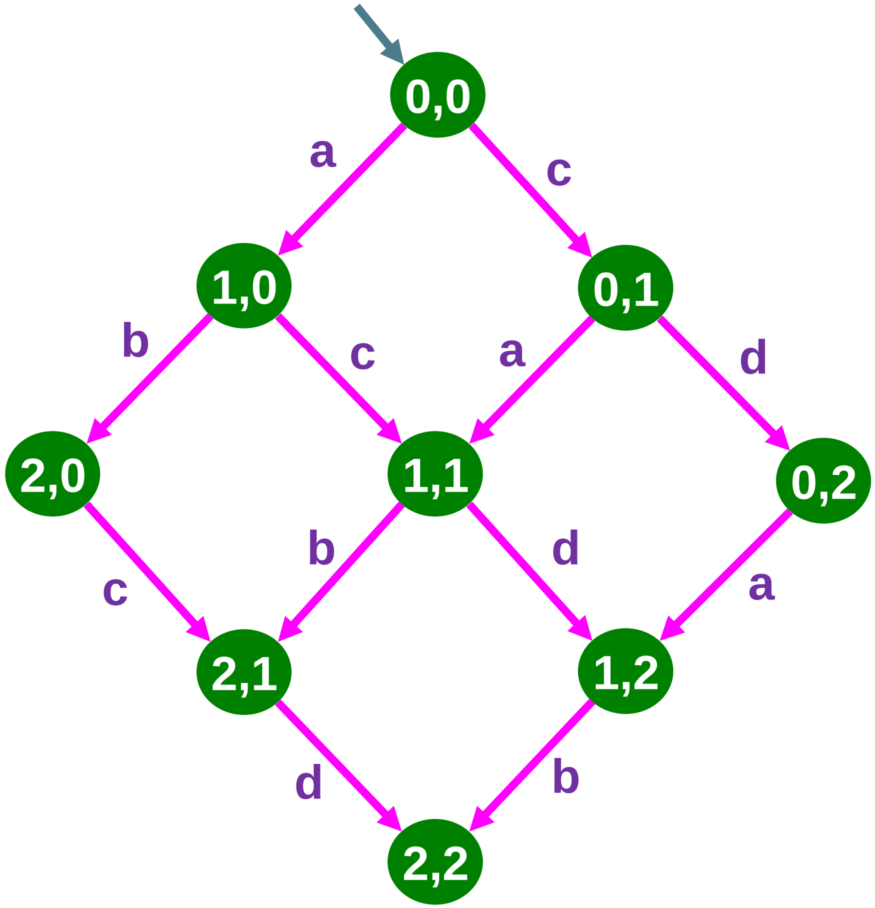



```
a >-> b  ||| c >-> d
```

Representing LTS







a >-> b

c >-> d

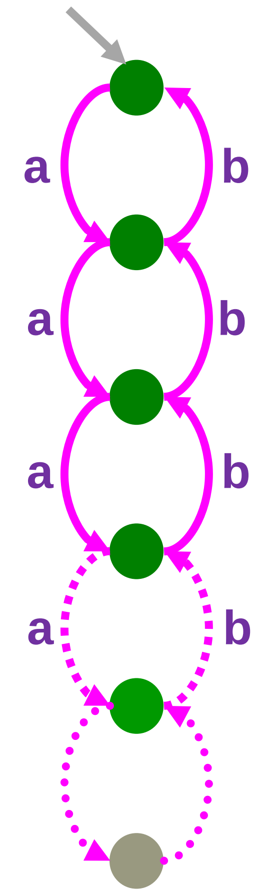
a >-> b ||| c >-> d

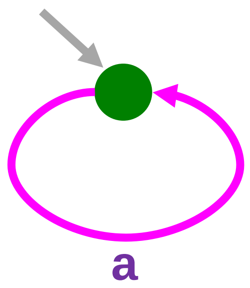
Representing LTS

Q

where

Q ::= a >-> (b ||| Q)





P

where

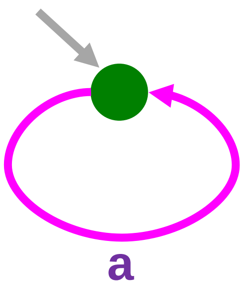
P ::= a >-> P

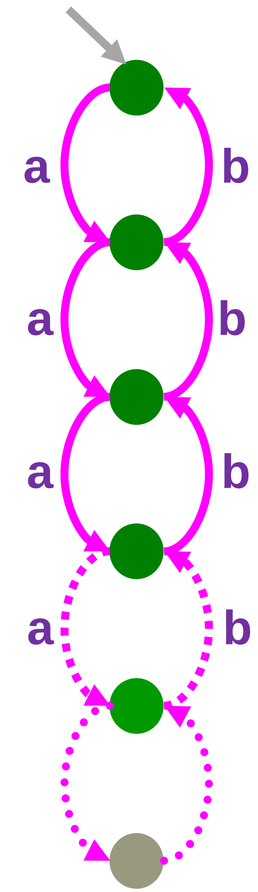
Representing LTS

Q

where

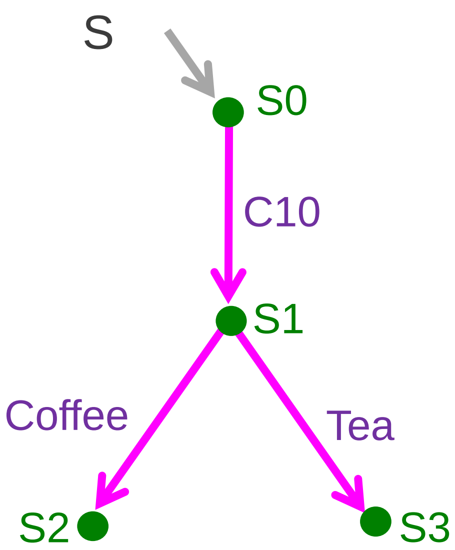
Q ::= a >-> (b ||| Q)






```
STAUTDEF P [..] (..)
 ::=
   STATE   p
   INIT    p
   TRANS  p  ->  a  ->  p
ENDDEF
```

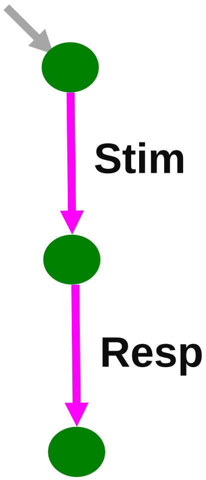
Language for LTS



- Explicit : ⟨ { S0, S1, S2, S3 } ,
  {10c,Coffee,Tea} ,
  { (S0, C10, S1),
  (S1, Coffee, S2),
  (S1, Tea, S3) } ,
  S0 ⟩

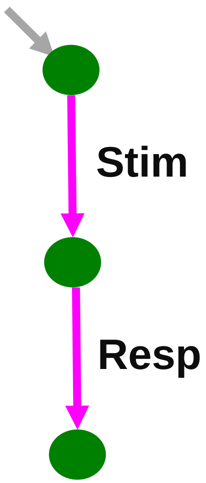
```
STAUTDEF  coffeeMachine  [ C10, Coffee, Tea ] ( )
   ::=
        STATE       S0, S1, S2, S3
        INIT        S0
        TRANS
                    S0  ->  C10      ->  S1
                    S1  ->  Coffee  ->  S2
                    S1  ->  Tea      ->  S3
ENDDEF
```

TorXakis: Behaviour Definition




```
MODELDEF  Mod
   ::=
            CHAN IN      Stim
            CHAN OUT   Resp

            BEHAVIOUR

                             Stim   >-> Resp

ENDDEF
```

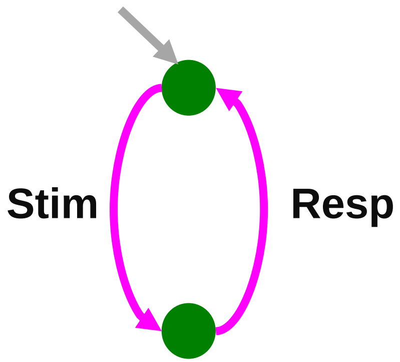
TorXakis: Process Definition




```
PROCDEF  stimResp  [ Stm, Rsp ]  ( )
   ::=
            Stm  >-> Rsp
ENDDEF
```

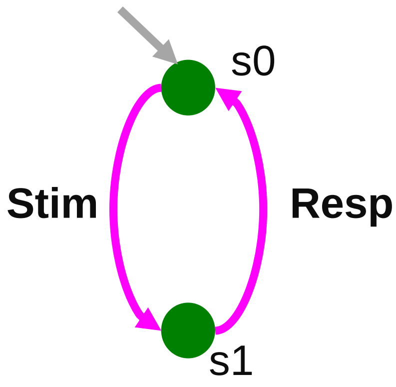
```
MODELDEF  Mod
   ::=
            CHAN IN          Stim
            CHAN OUT         Resp

            BEHAVIOUR

                     stimResp [ Stim, Resp ]  ( )

ENDDEF
```

TorXakis: Process Definition



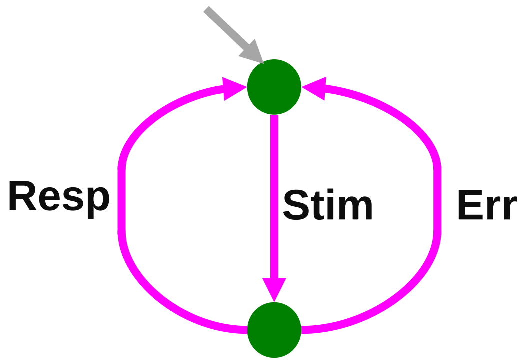

```
PROCDEF stimResp [ Stm, Rsp ] ( )
  ::=
                 Stm
           >-> Rsp
           >-> stimResp [ Stm, Rsp ] ( )
ENDDEF


MODELDEF Mod
  ::=      CHAN IN      Stim
           CHAN OUT   Resp

           BEHAVIOUR
                      stimResp [ Stim, Resp ] ( )
ENDDEF
```

TorXakis: Process Definition




```
STAUTDEF stimResp [ Stm, Rsp ] ( )
  ::=
          STATE   s0, s1
          INIT    s0
          TRANS   s0  ->  Stm  ->  s1
                  s1  ->  Rsp  ->  s0
ENDDEF


MODELDEF Mod
  ::=     CHAN IN      Stim
          CHAN OUT     Resp

          BEHAVIOUR
                  stimResp [ Stim, Resp ] ( )
ENDDEF
```

TorXakis: Choice




```
STAUTDEF errSR1 [ Stm, Rsp ] ( )
  ::=
          STATE   s0, s1
          INIT    s0
          TRANS   s0  ->  Stm  ->  s1
                  s1  ->  Rsp  ->  s0
                  s1  ->  Err  ->  s0
ENDDEF
```

```
--  Stimulus-Response with Error

PROCDEF  errSR  [ Stim, Resp, Err ]  ()
  ::=
       Stim   >->
          (   Resp  >->    errSR [Stim,Resp,Err] ()
             ##
              Err     >->    errSR [Stim,Resp,Err] ()
          )
ENDDEF
```

TorXakis

Data Definitions and Functions

TorXakis: Data Types

- Standard types: Int, Bool, String

- Algebraic data types

```
TYPEDEF  Colour  ::=  Red | Yellow | Blue  ENDDEF


TYPEDEF  IntList  ::=    Nil
                       | Cons  {  hd :: Int
                               ,  tl :: IntList
                               }
ENDDEF
```

TorXakis: Func

```
TYPEDEF   IntList    ::=      Nil
                            | Cons   {   hd  ::  Int
                                     ,   tl   ::  IntList
                                     }
ENDDEF
```

- Functions: name, pa
- Overloading
- Standard functions for: Int, Bool, String

```
FUNCDEF   ++  ( s :: IntList;  x :: Int )  ::  IntList
   ::=
         IF    isNil ( s )
         THEN   Cons ( x, Nil )
         ELSE   Cons ( hd ( s ),  tl ( s ) ++ x )
         FI
ENDDEF
```

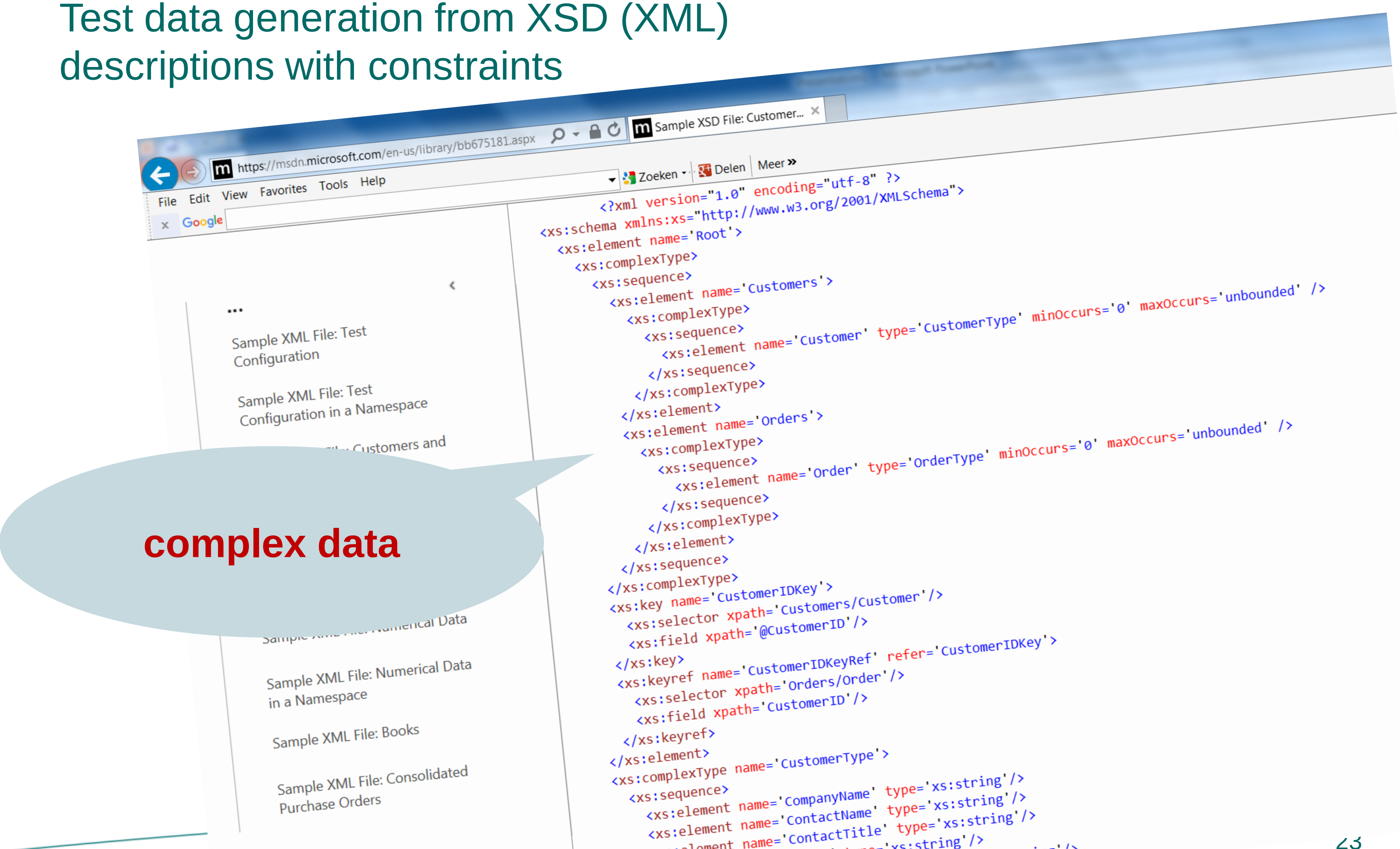
TorXakis: Data Types

```
TYPEDEF IntStringMap
 ::=
      NoMap
     | Map  { index :: Int
             ; value  :: String
             ; rest    :: IntStringMap
             }
ENDDEF

FUNCDEF lookup ( i :: Int; map :: IntStringMap ) :: String
 ::=
      IF       isNoMap(map)
     THEN ""
     ELSE IF       index(map) == i
             THEN  value(map)
             ELSE  lookup(i,rest(map))
             FI
     FI
ENDDEF
```

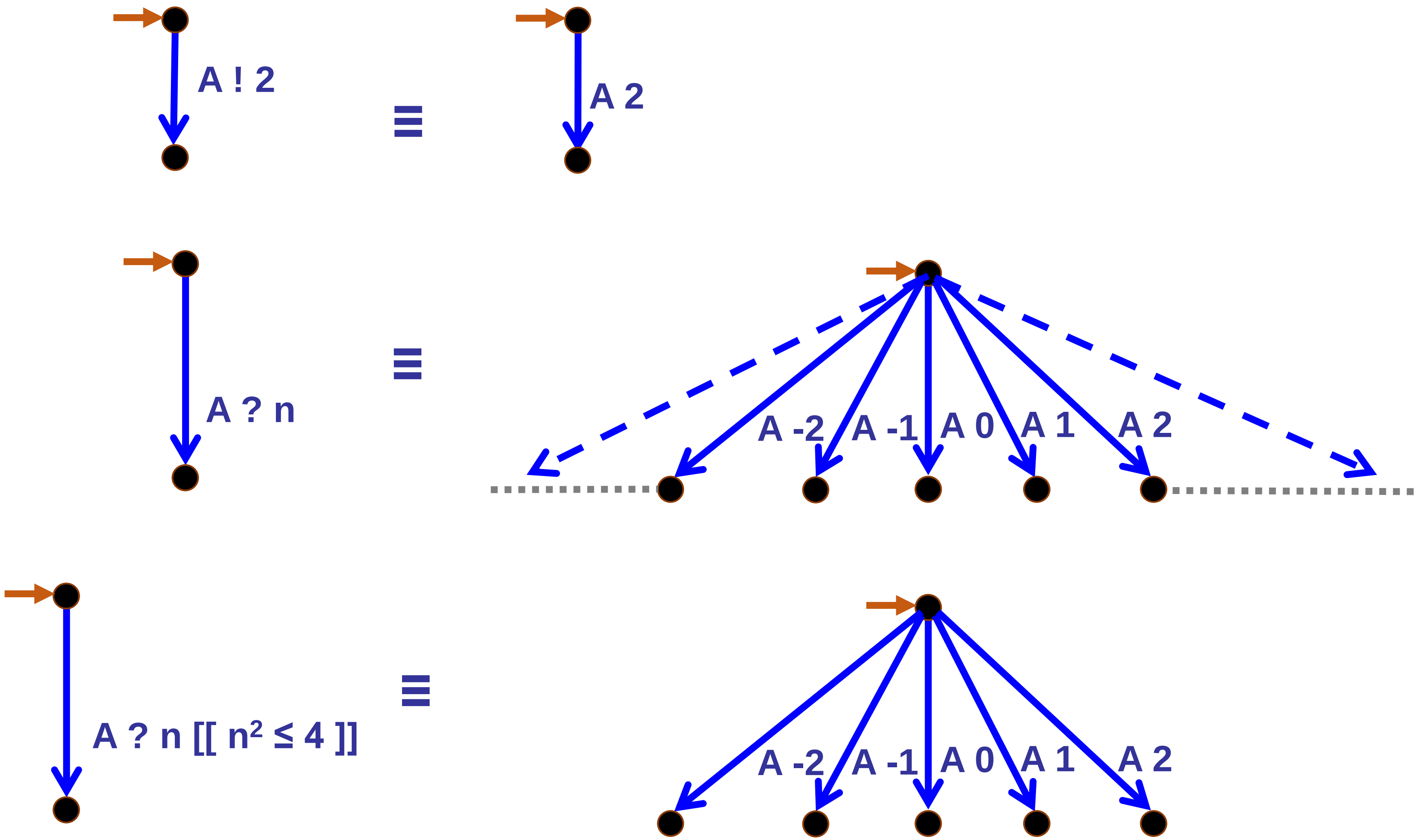
```
CONSTDEF someMap :: IntStringMap
 ::=
     Map(1,"Aap",
     Map(2,"Noot",
     Map(3,"Mies",NoMap)))
ENDDEF
```

More Complex Data

Test data generation from XSD (XML) descriptions with constraints



TorXakis : LTS with Data = STS (*Symbolic Transition System*)



TorXakis: Adder with Data

Action ? opn :: Operation

Result ? n :: Int

```
TYPEDEF  Operation
  ::=    Plus  { p1, p2 :: Int }
       |  Minus { m1, m2 :: Int }
       |  Error
ENDDEF
```

```
PROCDEF  adder  [ Act :: Operation;  Res :: Int ] ( )
   ::=
              Act ?opn [[ isPlus(opn)  ]]
        >->  Res !p1(opn)+p2(opn)
        >->  adder [ Act, Res ] ( )
    ##
              Act ?opn [[ isMinus(opn) ]]
        >->  Res !m1(opn)-m2(opn)
        >->  adder [ Act, Res ] ( )
ENDDEF
```

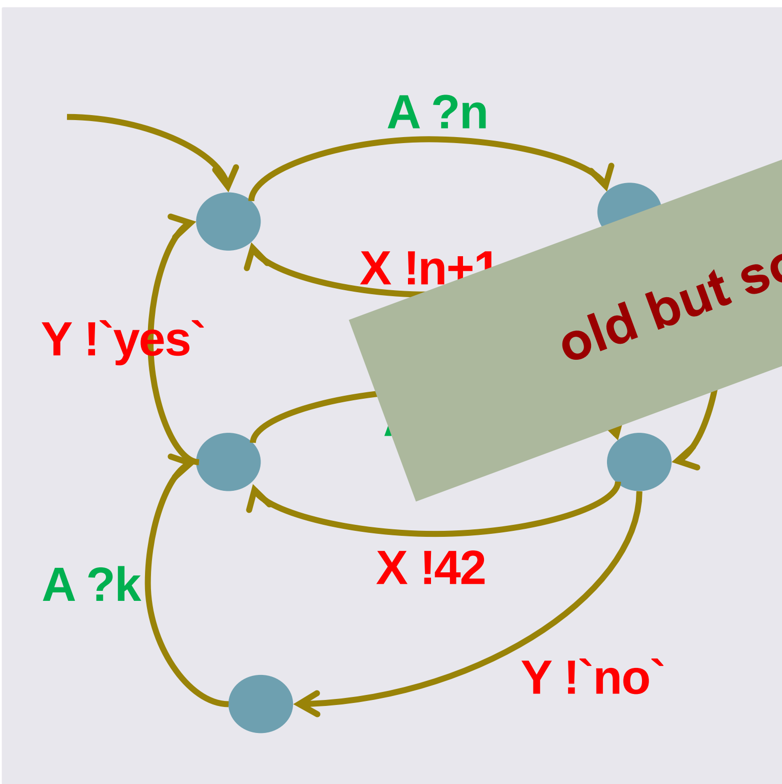
TXS

More Models

Language for Composition

TorXakis : Defining Behaviour - LTS

basic behaviour
= transition system

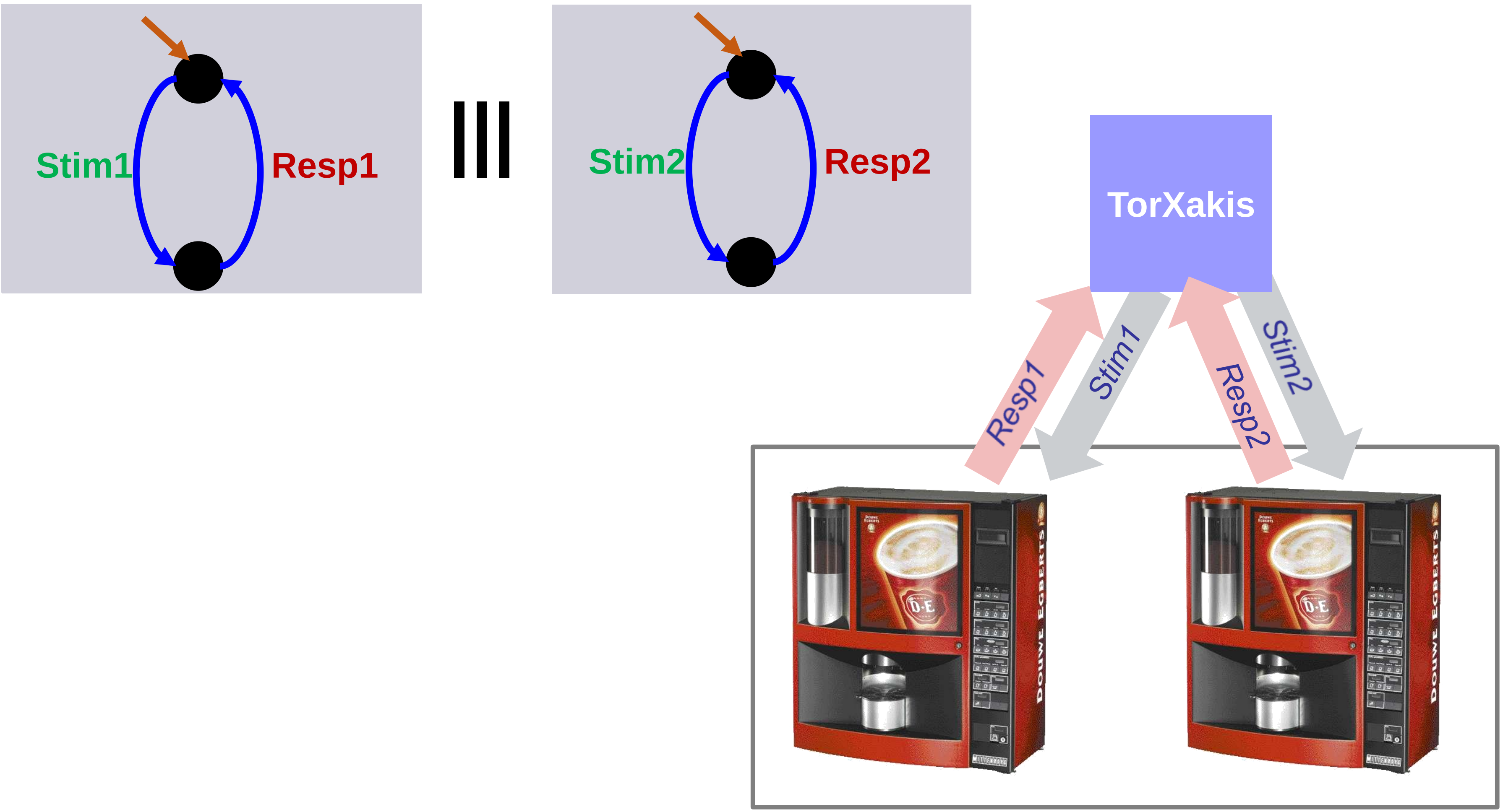


complex behaviour
= combining transition systems

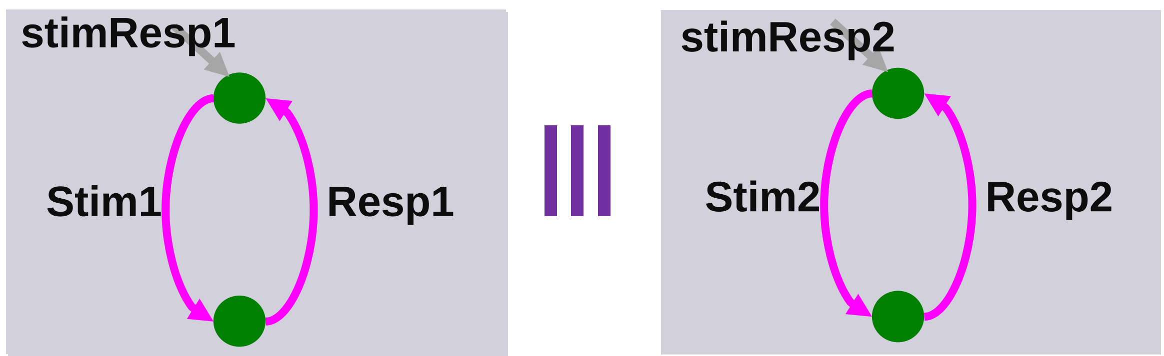
- name… haviour definition
- …aviour use
- …
- …hoice
- parallel
- communication
- exception
- interrupt
- hiding

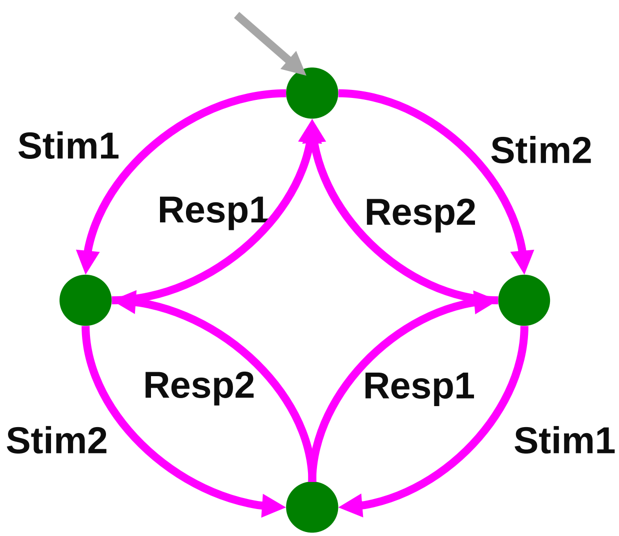
old but solid theory

TorXakis: Parallel Interleaving



TorXakis: Parallel Interleaving

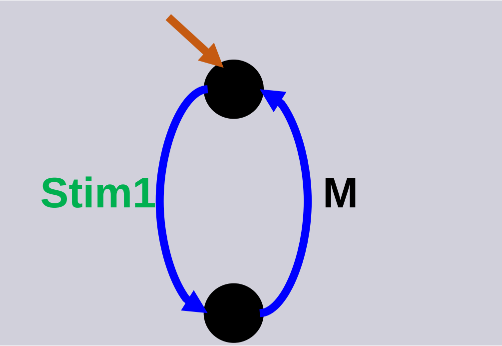




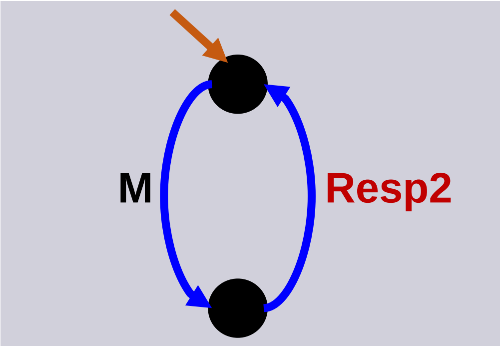
Parallelism with interleaving:

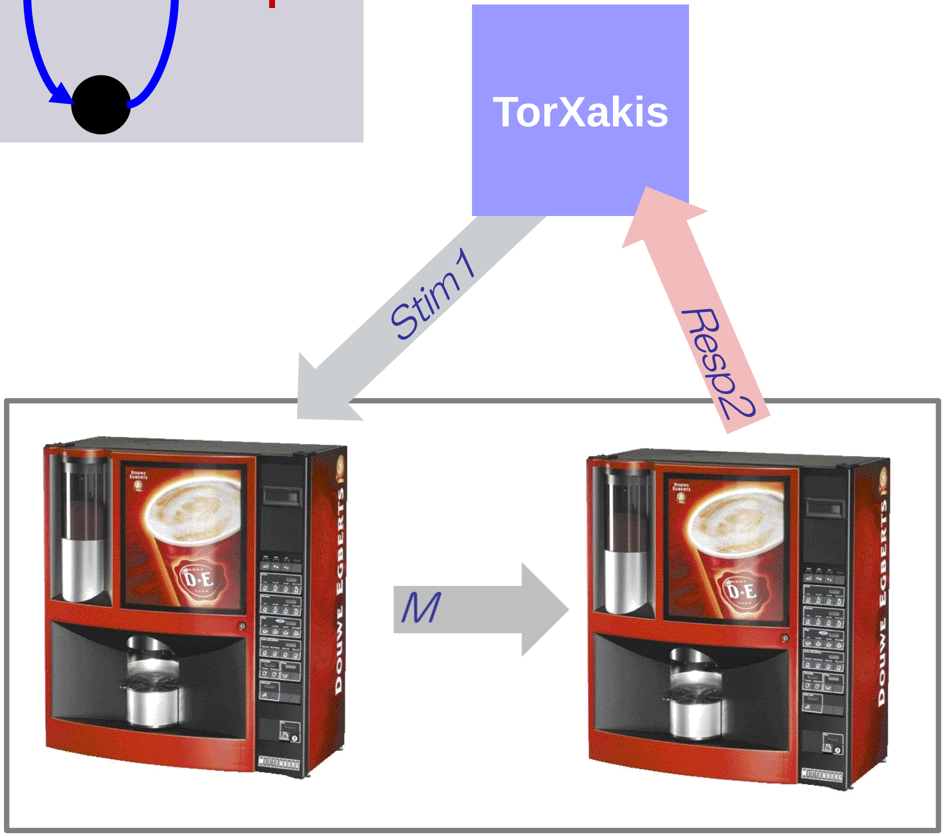
stimResp1 ||| stimResp2

TorXakis: Parallel Communication

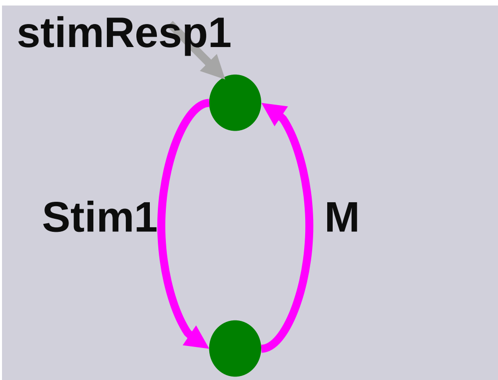


|[M]|

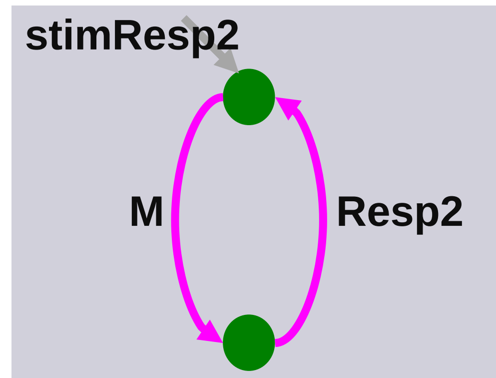


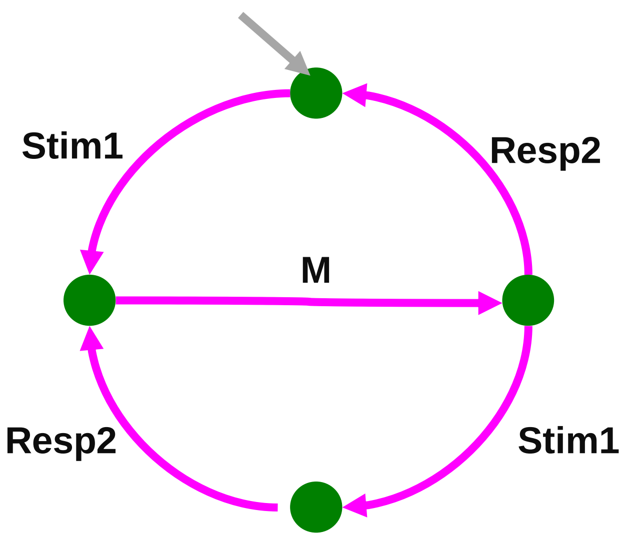


TorXakis: Parallel Communication



|[M]|

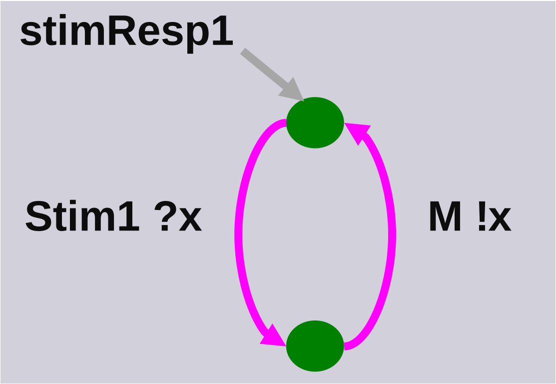




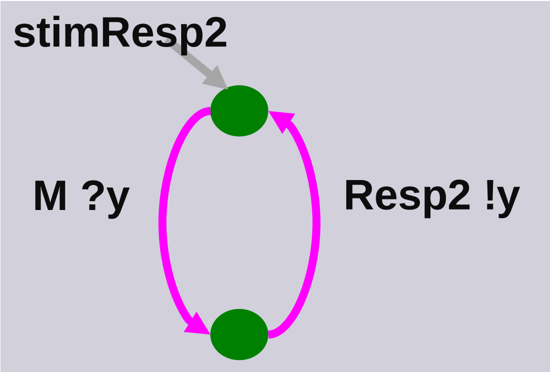
Parallelism with communication:

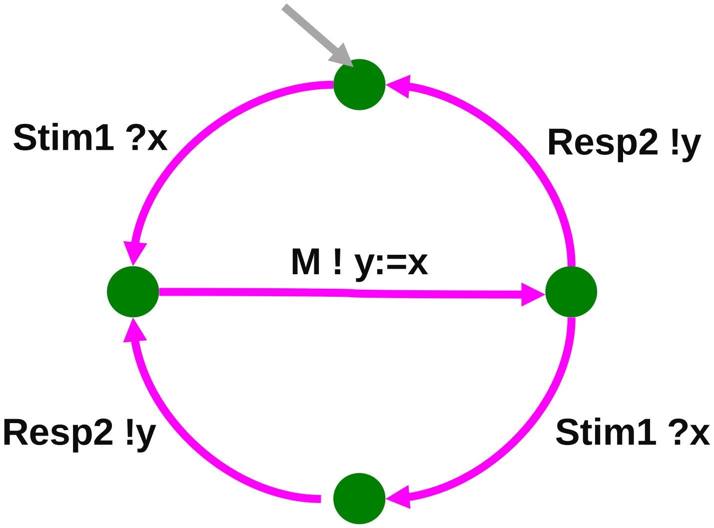
stimResp1 |[M]| stimResp2

TorXakis: Parallel Communication



|[M]|

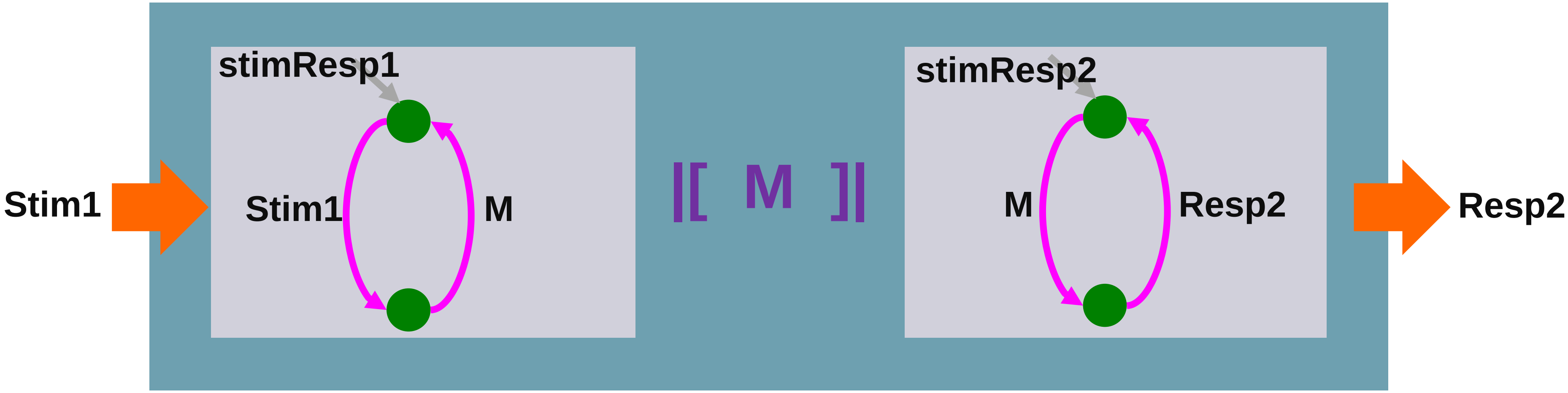


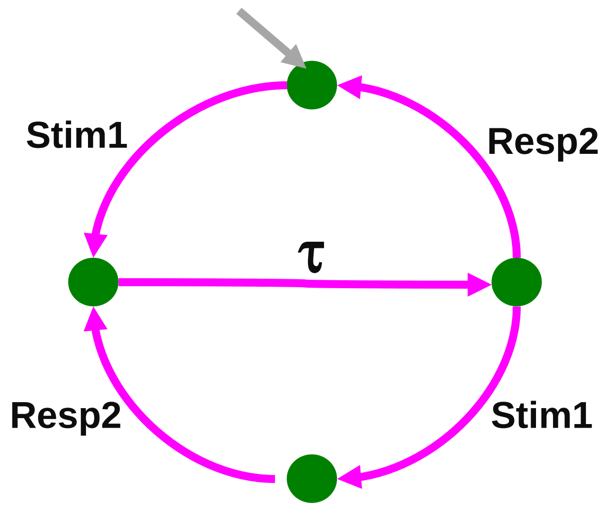


Parallelism with communication:

stimResp1 |[M]| stimResp2

TorXakis: Communication + Hiding (Abstraction)

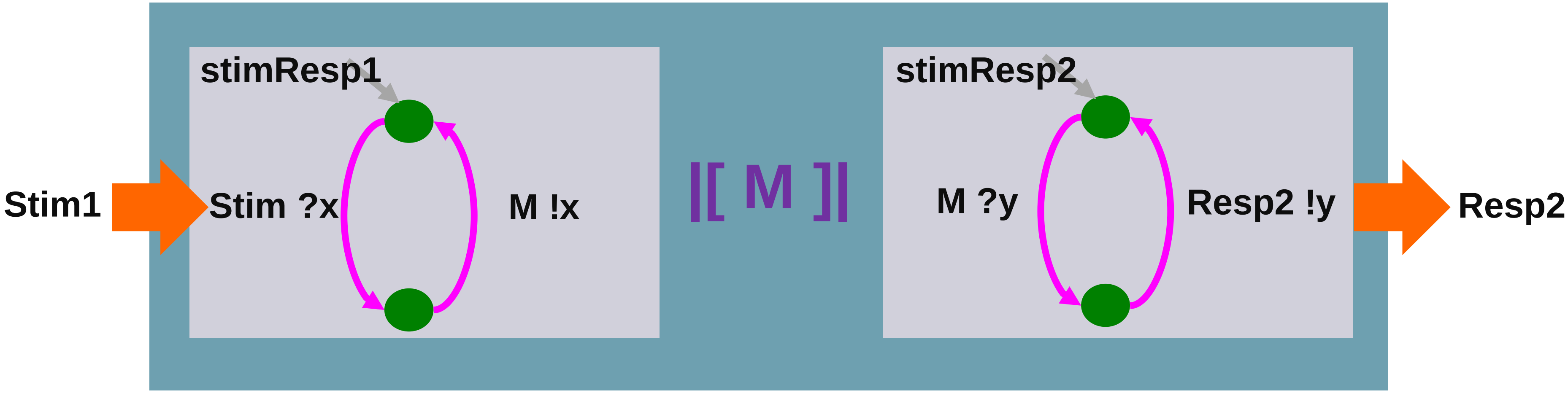


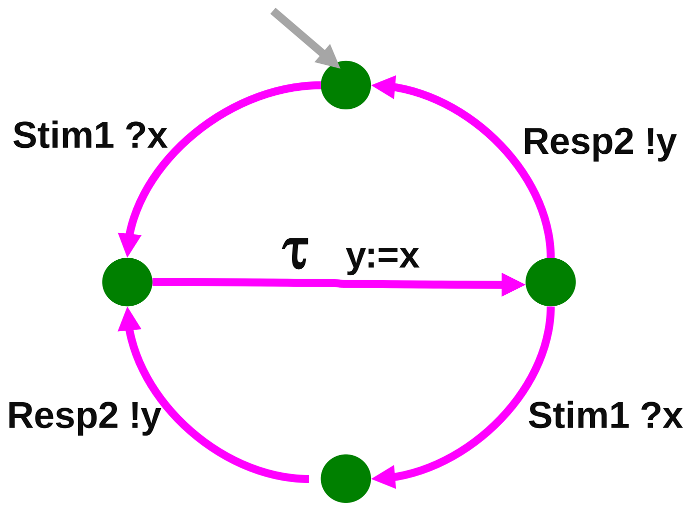


Communication + Hiding:

HIDE [M]
IN
stimResp1 |[M]| stimResp2
NI

TorXakis: Communication + Hiding (Abstraction)





Communication + Hiding:

HIDE [M]
IN
stimResp1 |[M]| stimResp2
NI

TorXakis: Behaviour Compositions

Enable	**proc1** **>>>** **proc2**	*when* proc1 *finishes,* proc2 *continues*
Disable	**proc1** **[>>** **proc2**	*the first action of* proc2 *disables* proc1
Interrupt	**proc1** **[><** **proc2**	*the first action of* proc2 *disables* proc1; *when* proc2 *finishes,* proc1 *continues where it stopped*

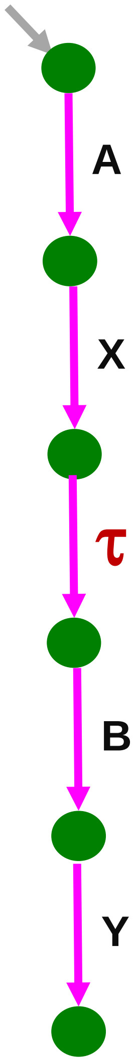
TorXakis: Enable

proc1 >>> proc2

when proc1 *finishes with* **EXIT**,
then proc2 *continues*

proc1 ::= A >-> X >-> EXIT

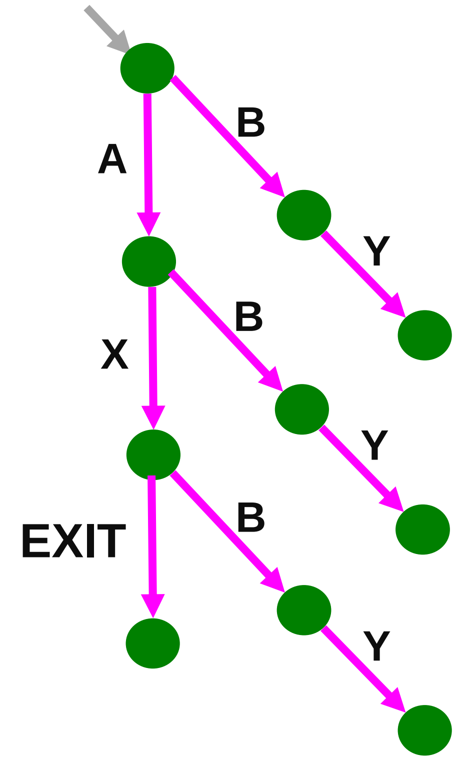
proc2 ::= B >-> Y



TorXakis: Disable

proc1 [>> proc2

the first action of proc2
disables proc1
except if proc1 *finished*
with **EXIT**



proc1 ::= A >-> X >-> EXIT

proc2 ::= B >-> Y

TorXakis: Interrupt

proc1 [>< proc2

the first action of proc2
disables proc1
except if proc1 *finished with* **EXIT***;*
when proc2 *finishes with* **EXIT***,*
proc1 *continues where it stopped,*
and can be interrupted again

proc1 ::= A >-> X >-> EXIT

proc2 ::= B >-> Y >-> EXIT