

# Modeling the Exogenous Coordination of Mobile Channel based Systems with Petri Nets

**Juan Guillen-Scholten**

CWI

Internal Slides, September, 2005.



# Overview



- ✓ Our Goal.
- ✓ The MoCha Framework & Middleware.
- ✓ Elementary Net Systems (an introduction).
- ✓ Modeling Systems
- ✓ Defining a Protocol.
- ✓ Composition
- ✓ Modeling Channels
  - Synchronous
  - FIFO-1
  - Asynchronous Drain
  - Lossy Synchronous
- ✓ Analysis and Simulation.
- ✓ Complexity of Our Approach.
- ✓ Tool Example.
- ✓ Summary and Conclusions.

# Our Goal

---

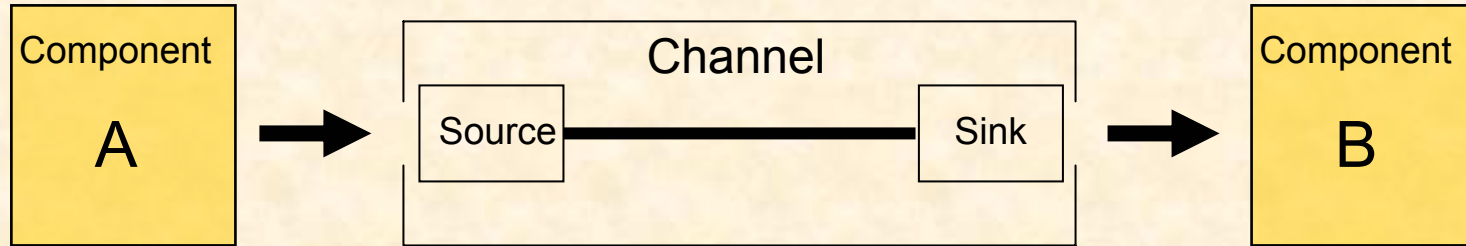
To model the exogenous coordination behavior of mobile channel based systems.

We use Petri Nets as our modeling language so we can take advantage of all the theory and tools available for them:

- (1) PNs are widely used in both academia as well as in industry
- (2) They have well-defined semantics with clear theoretical foundation.
- (3) They provide analysis.
- (4) They provide simulation.
- (5) PN's are easy to understand.
- (6) There is extensive **tool support** for PN.

# The MoCha Framework (1)

---

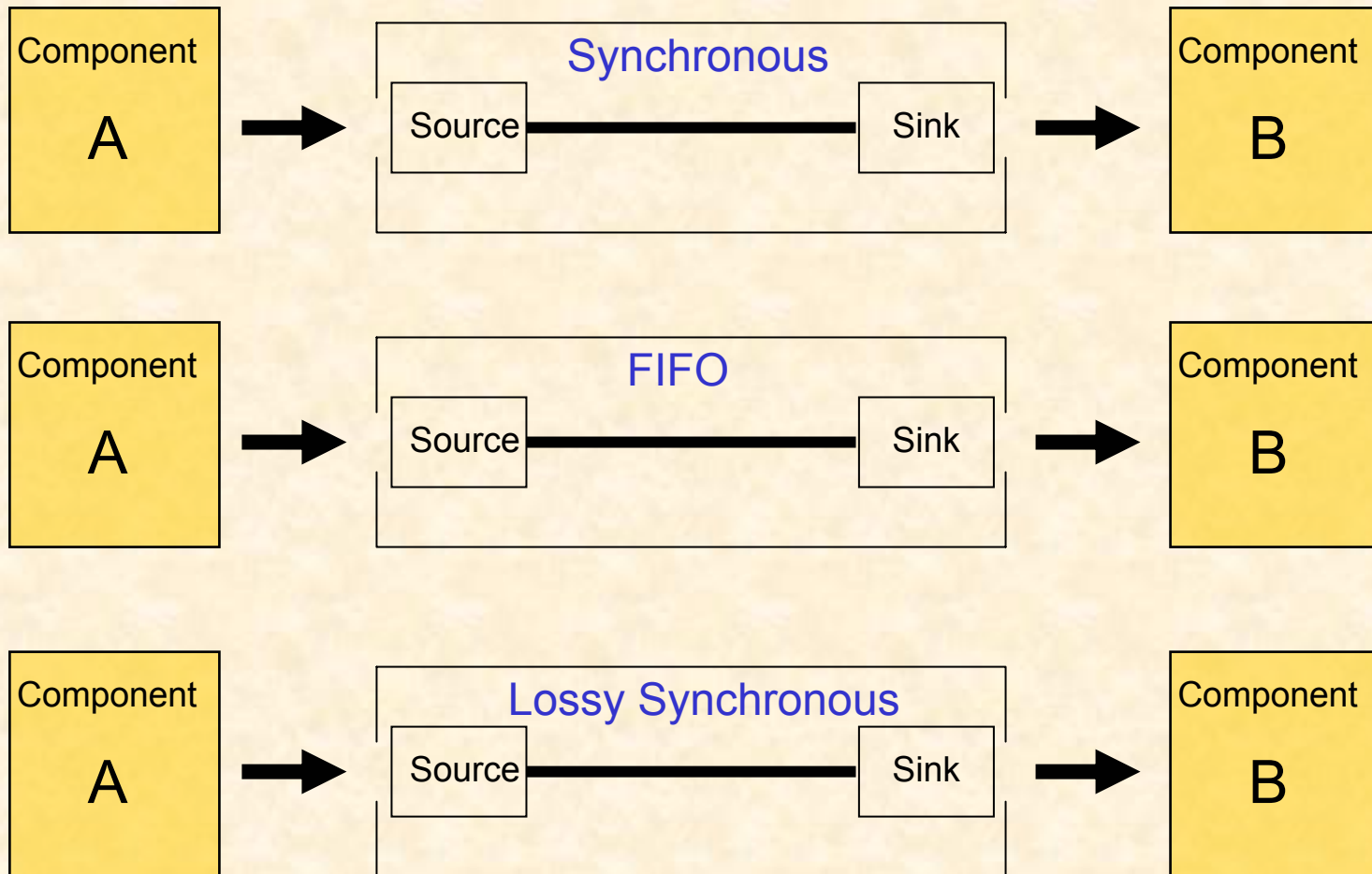


- A channel has two distinct ends: Usually (*source, sink*), *but also (source, source) and (sink, sink)*.
- Components write to the source-end, and take from the sink-end.
  - ✓ The I/O operations performed on channel-ends are synchronous.
- The communication is *anonymous*.
- Channels have different types (synchronous and asynchronous).
- Channels are mobile **➡** Their channel-ends are mobile.

# The MoCha Framework (2)

---

✓ *Transparent exogenous coordination.*

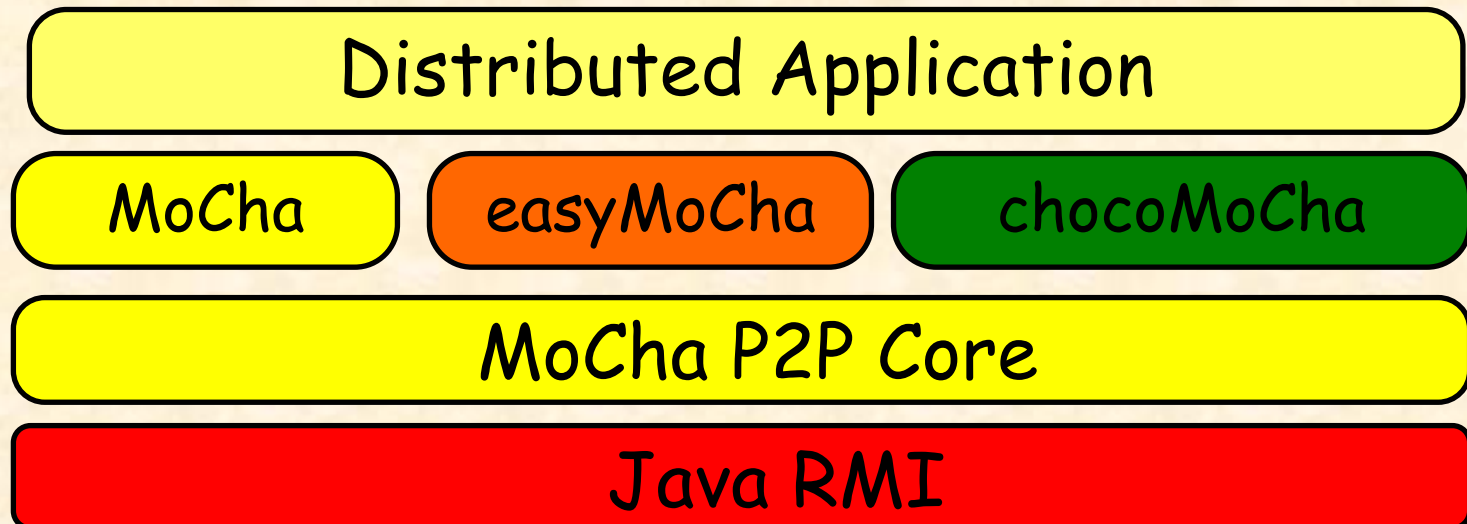


# The MoCha Middleware

---

The *MoCha middleware*: Implementation of the Framework in Java RMI.

- Fully decentralized, Peer-to-Peer architecture.
- Comes in different flavors: *MoCha*, *easyMoCha*, and *chocoMoCha*.



# Choice of Petri Net Model

---

- High-level PNs: *Colored Petri Nets*, *Predicate/Transition Nets*.
- Intermediate-level PN: *Place/Transition Systems* (P/T systems).
- Low-level PN: *Elementary Net Systems* (EN systems).

Both P/T and EN systems have clear non-changeable semantic rules and constructs. This is easier for analysis and simulation (especially when using tools).

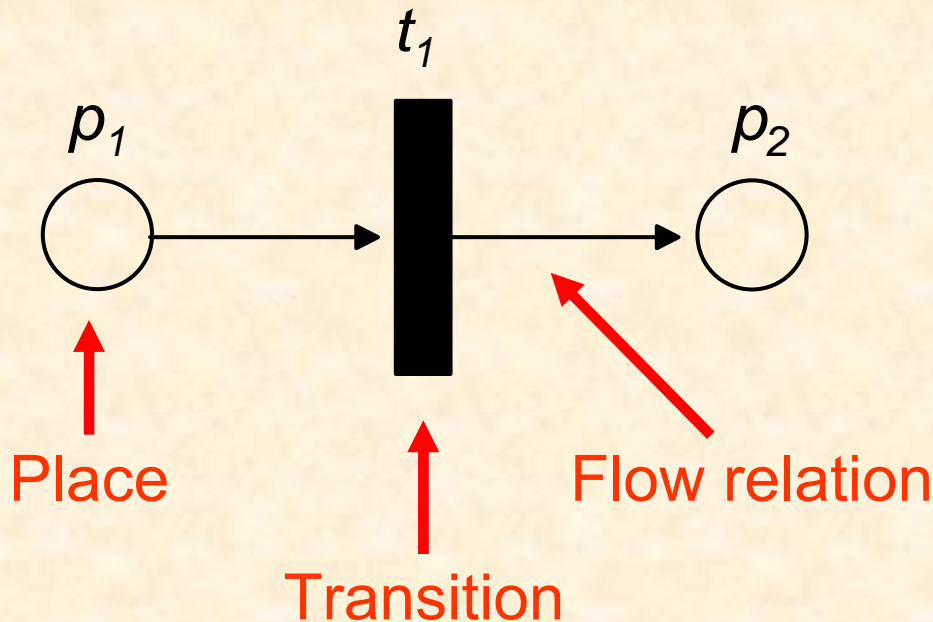
We have modeled everything using P/T and EN systems. In this talk, and in the paper, we use the *EN systems* PN.

# Elementary Net Systems (1)

---

Def: A net is triple  $N = (P, T, F)$ , where:

- 1)  $P$  and  $T$  are finite sets with  $P \cap T = \emptyset$ ,
- 2)  $F \subseteq (P \times T) \cup (T \times P)$ ,
- 3) for every  $t \in T$  there exists  $p, q \in P$  such that  $(p, t), (t, p) \in F$ , and
- 4) for every  $t \in T$  and  $p, q \in P$ , if  $(p, t), (t, p) \in F$ , then  $p \neq q$ .



$$P = \{p_1, p_2\},$$

$$T = \{t_1\},$$

$$F = \{(p_1, t_1), (t_1, p_2)\}$$

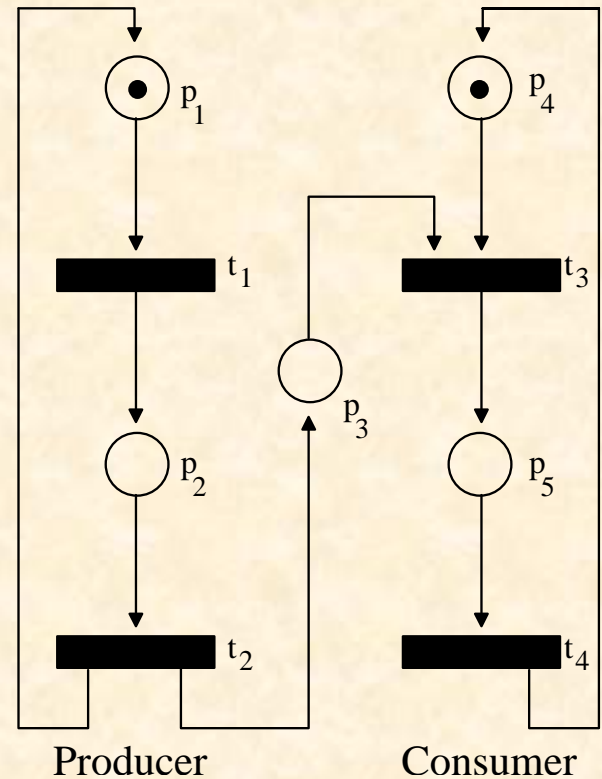
# Elementary Net Systems (2)

Def: A *configuration* of a net is a subset of  $P$ , where each place contains a token.  
(A token in a place denotes the fact that it participates in the current global state).

Def: An EN system is a quadruple

$M = (P, T, F, Cin)$  where:

- 1)  $(P, T, F)$  is a net, and
- 2)  $Cin \subseteq P$  is the *initial configuration*.



$$Cin = \{p_1, p_4\}$$

# Elementary Net Systems (3)

---

Def. Firing:

$M = (P, T, F, C_{in})$  and  $t \in T$ :

- 1)  $\bullet t$  are the input places of  $t$ , and  $t^\bullet$  the output places.
- 2) Let  $C$  be a configuration. Then  $t$  can be fired in  $C$ , if  $\bullet t \subseteq C$  and  $\bullet t \cap C = \emptyset$ , written as " $t \mathbf{con} C$ ".
- 3) Let  $C$  and  $D$  be configurations. Then  $t$  fires from  $C$  to  $D$ , if  $t \mathbf{con} C$  and  $D = (C - \bullet t) \cup t^\bullet$ , written as  $C[t>D$ .

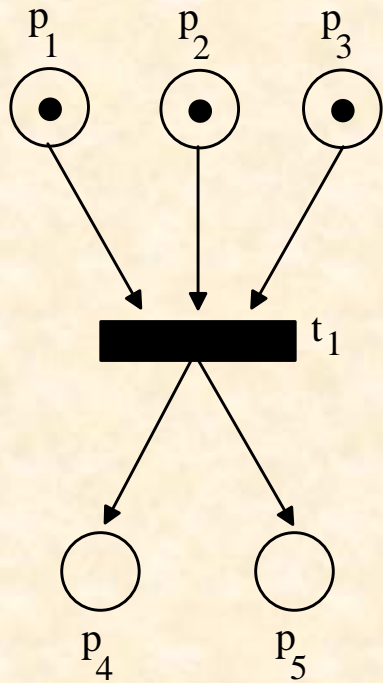
Informal: *A transition  $t$  can fire*

- 1) *if all input places of  $t$  contain tokens, and,*
- 2) *all output places of  $t$  are empty*

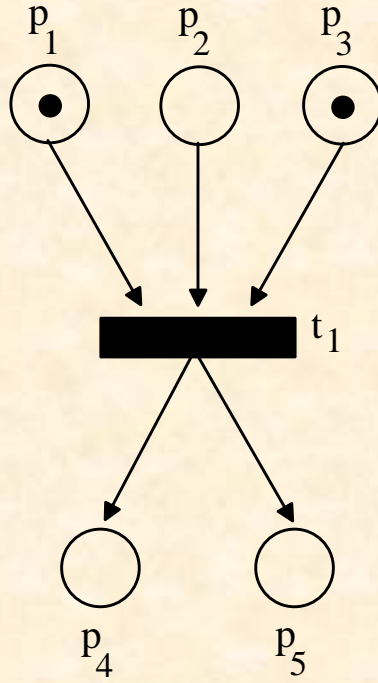
# Elementary Net Systems (4)

---

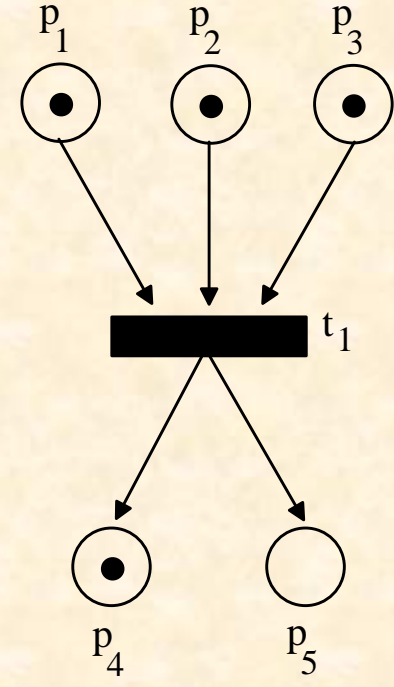
Examples:



Yes



No



No

$\{p_1, p_2, p_3\}[t_1] \rightarrow \{p_4, p_5\}$

# Modeling Systems

---

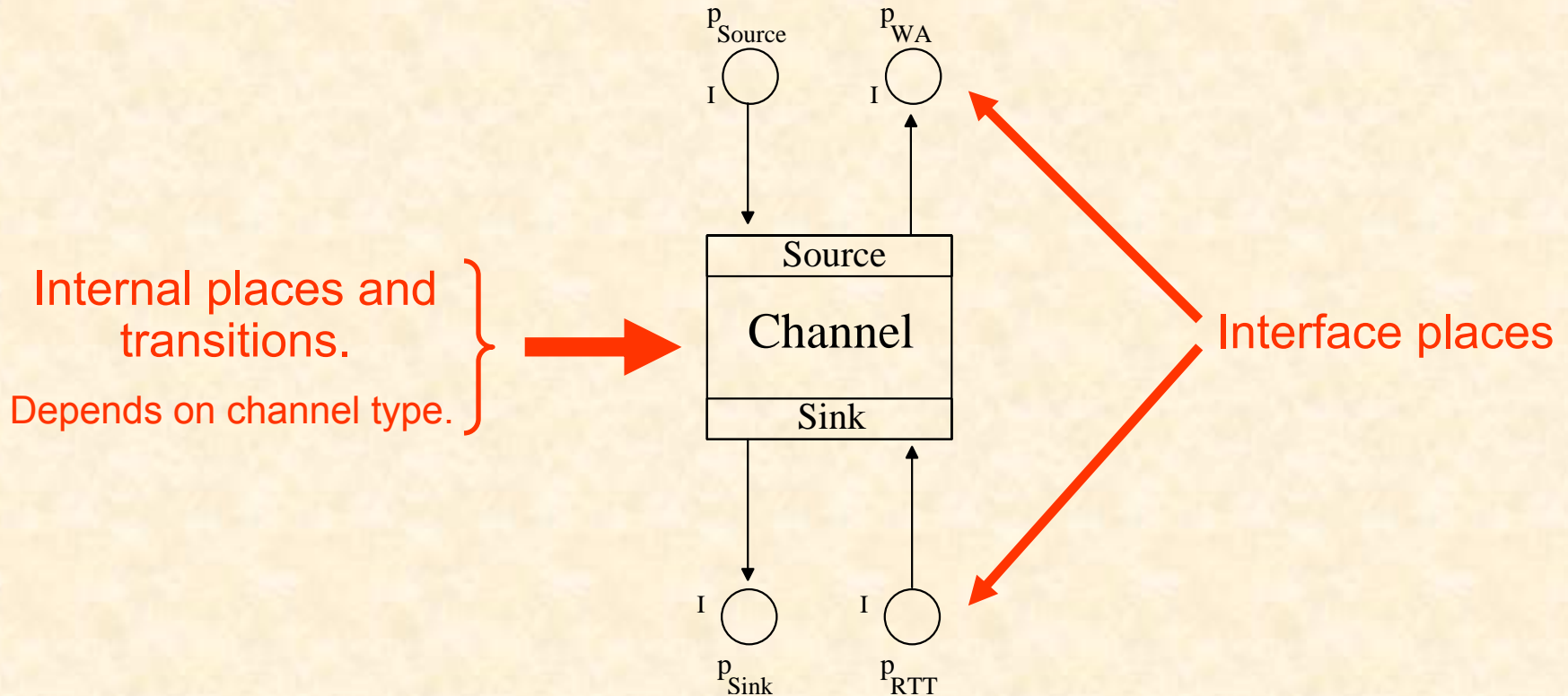
Our approach:

- 1) Give PN for a set of representative MoCha channel types.  
[all implemented in the middleware]
- 2) Discuss a small protocol for interaction between components and channels, that ensures that all operations between them are *blocking*.
- 3) Give a construction of components and channels that is *compositional*.

The order of the presentation is 2), 3) and 1).

# Protocol: Mobile Channel Interface (1)

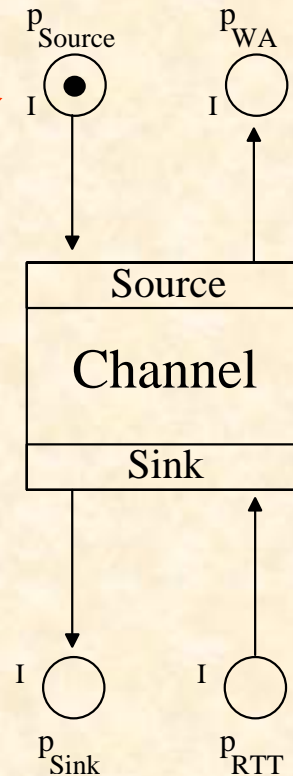
## PN Mobile Channel Interface:



# Protocol: Mobile Channel Interface (2)

Protocol, write operation:

A component starts a write operation, by inserting a token in

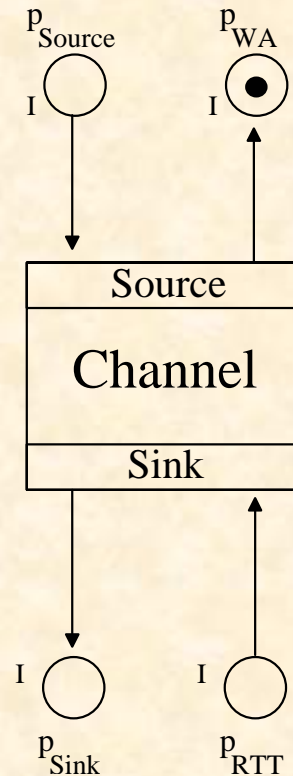


At some point the channel accepts the write and takes the token "inside".

# Protocol: Mobile Channel Interface (3)

Protocol, write operation:

The component needs to take this token before the operation is finished at his side.

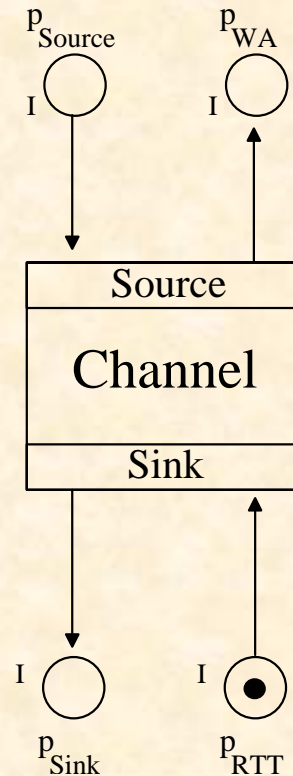


When the channel finish the write operation, it puts a token in the acknowledgement place

# Protocol: Mobile Channel Interface (4)

Protocol, take operation:

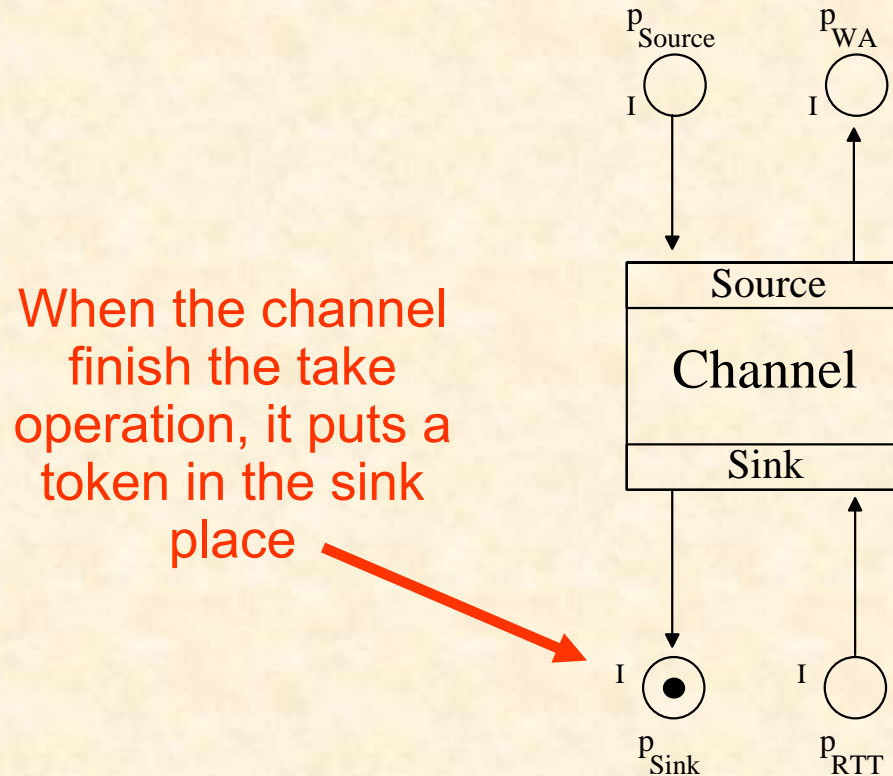
At some point the channel takes the token “inside” and knows that a component wants to take.



A component starts a take operation by inserting a token in the “Ready to Take” place

# Protocol: Mobile Channel Interface (5)

Protocol, take operation:



When the channel finish the take operation, it puts a token in the sink place

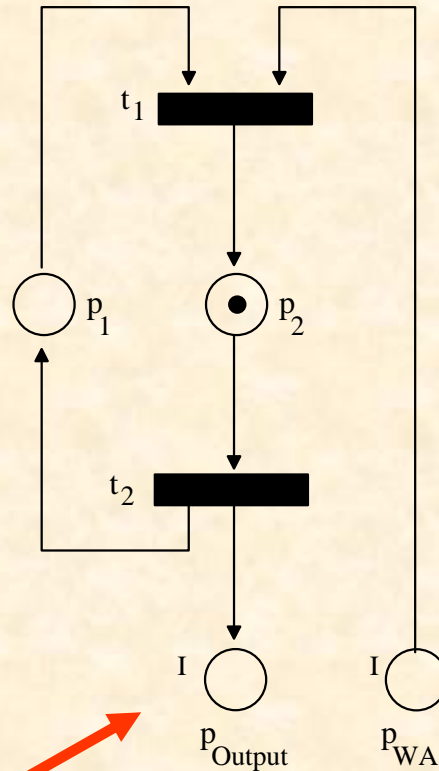
The component needs to take this token before the operation is finished at his side.

# Protocol: Components (1)

---

Protocol, writer:

Minimal behavior  
towards a write  
operation.



The active entity of  
the component  
blocks until it gets an  
acknowledgment  
back.

Coincides with the  
 $P_{source}$  place of the  
channel.

Coincides with the  
 $P_{wa}$  place of the  
channel.

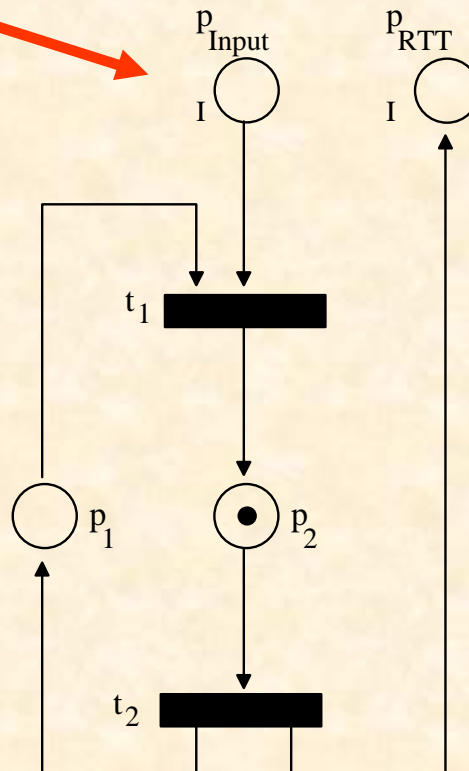
# Protocol: Components (2)

Protocol, taker:

Coincides with the  $P_{\text{sink}}$  place of the channel.

Coincides with the  $P_{\text{RTT}}$  place of the channel.

Minimal behavior towards a take operation.



The active entity of the component blocks until it gets a value back.

# Composition (1)

---

Construct PN systems by composing channels and components together.

Such a construction needs to be **compositional**:

- ✓ Always able to recognize the individual channels and components.
- ✓ Must be easy to decompose the systems into the original parts.
- ✓ No optimization allowed, otherwise we cannot recognize the individual constituents anymore.

Choice: compose on places or transitions.

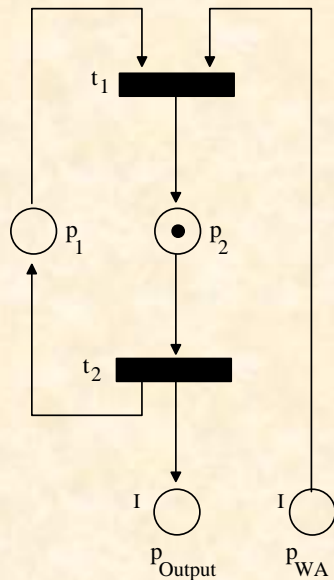
- composition on transitions may lead to invalid EN systems.

+ therefore, we do composition on (interface) places.

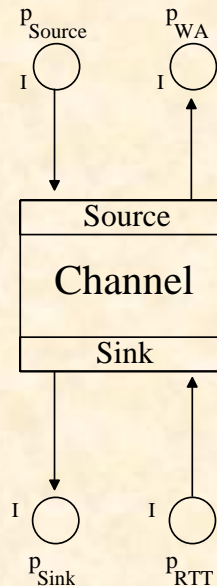
# Composition (2)

We define a composition function  $\sigma: (X_1, P_1, X_2, P_2) \rightarrow Y$ . Where  $X_1, X_2$  and  $Y$  are EN systems,  $P_1$  and  $P_2$  are selected places of resp.  $X_1$ , and  $X_2$  for composition.

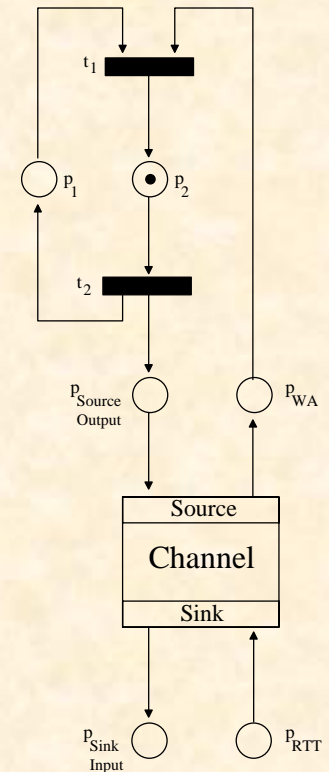
See the paper for technical details, here an example:



**X1**

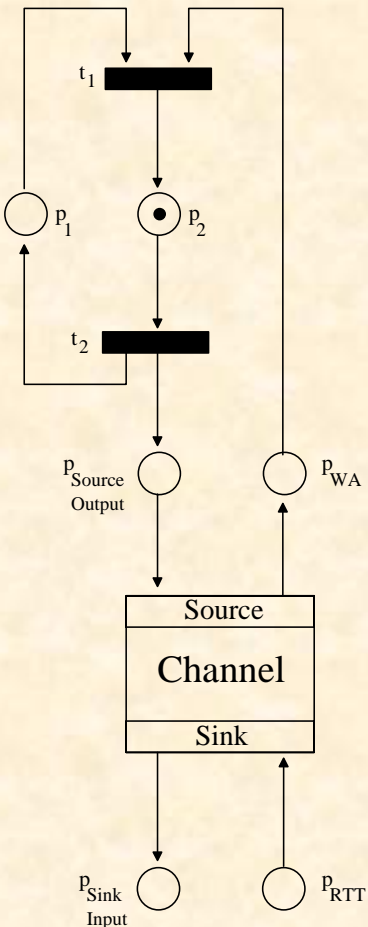


**X2**

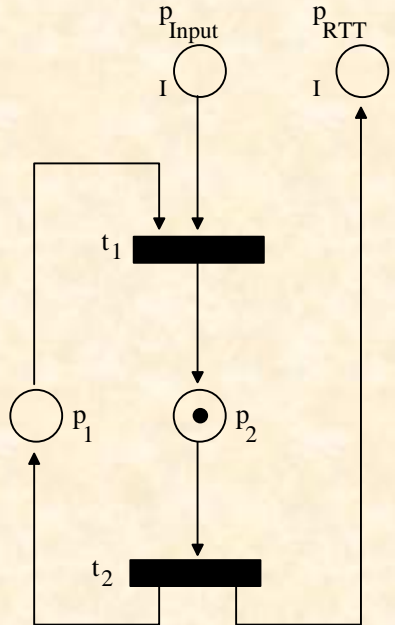


**Y**

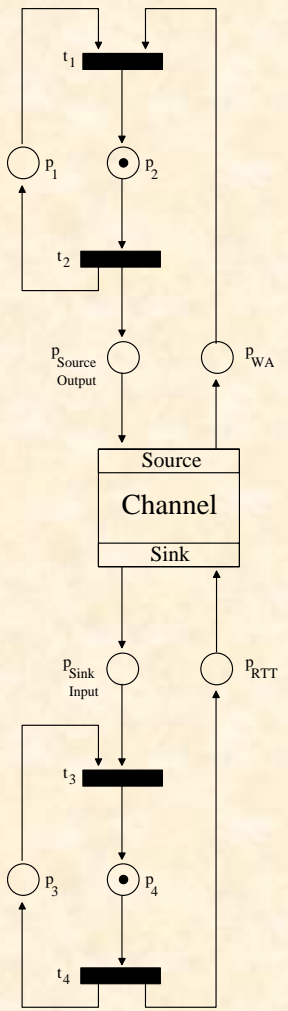
# Composition (3)



X1



X2



Y

# Modeling Channels

---

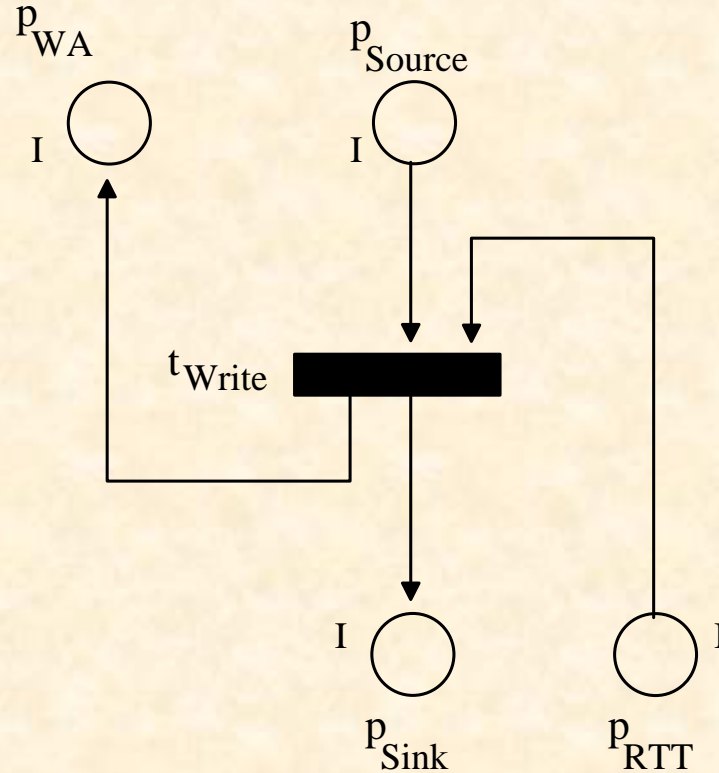
We now give a PN for a set of representative MoCha channel types:

- ✓ Synchronous
- ✓ FIFO-1
- ✓ Asynchronous Drain
- ✓ Lossy Synchronous

[all implemented in the middleware]

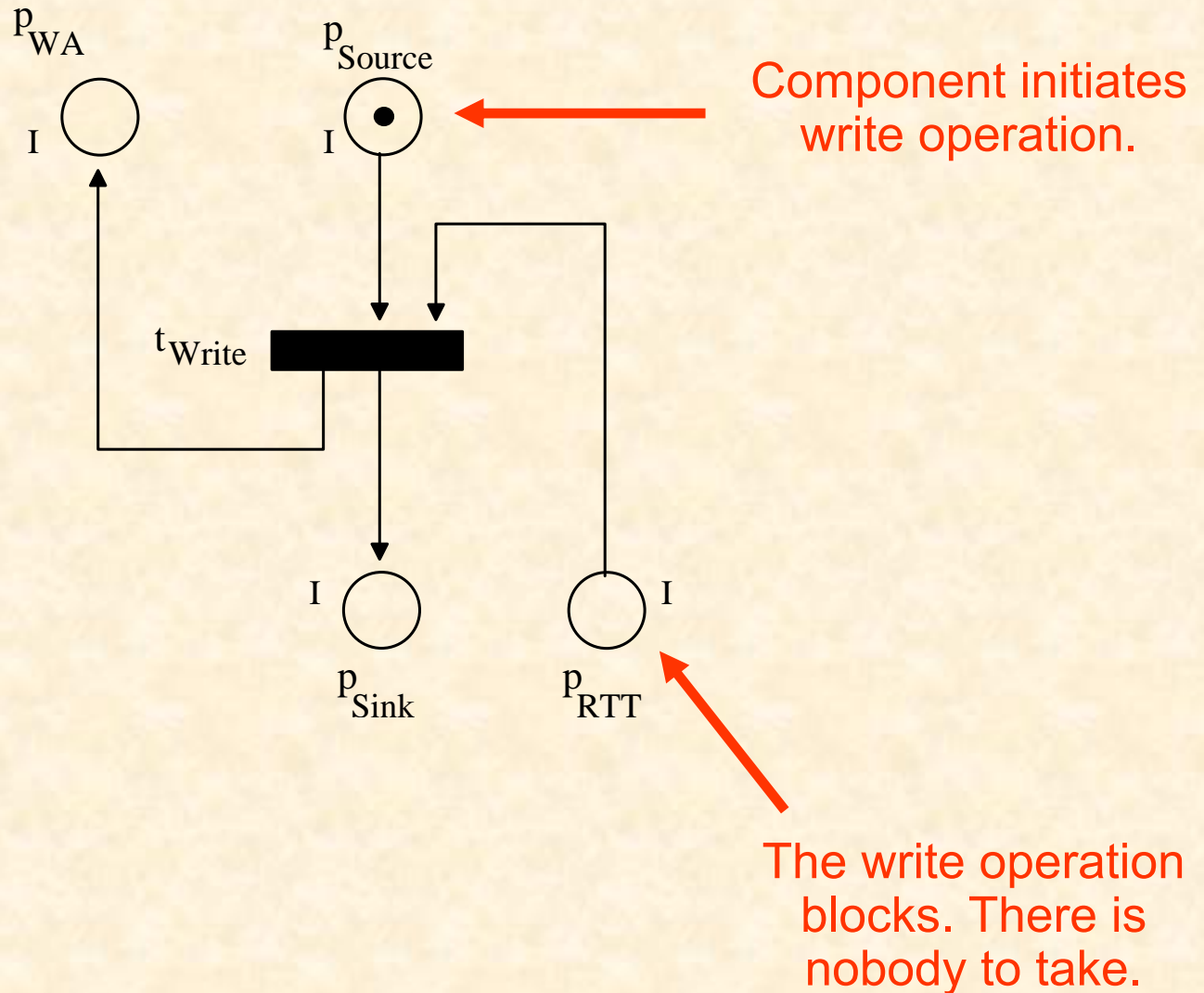
# Synchronous Channel(1)

---



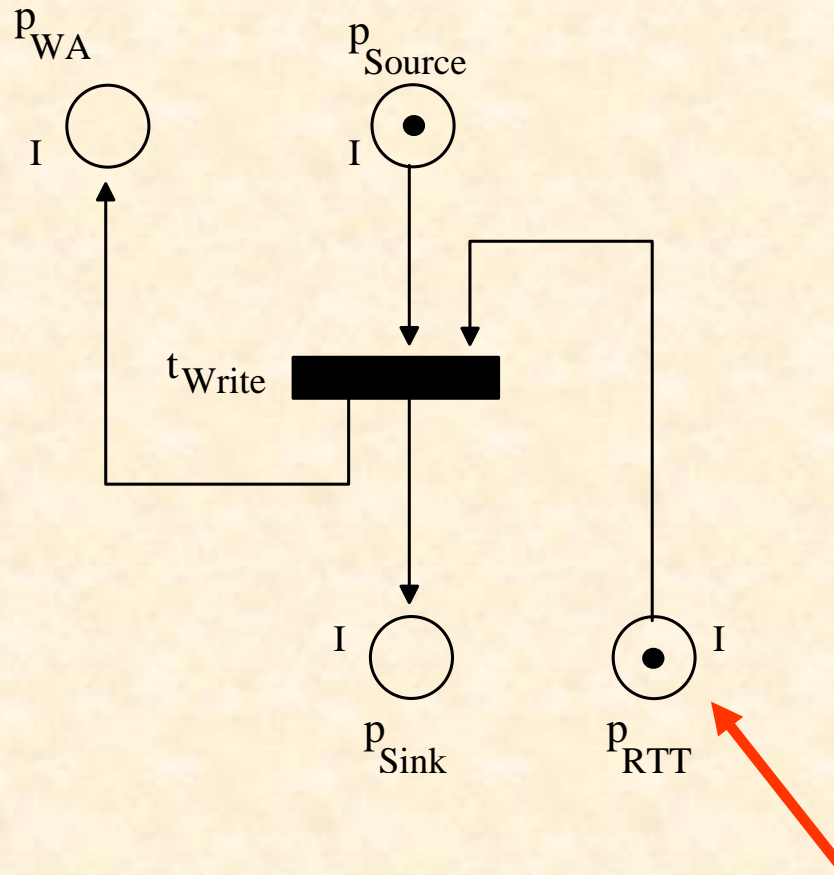
The I/O operations of both ends are synchronized; i.e. the I/O operations atomically succeed.

# Synchronous Channel(2)



# Synchronous Channel(3)

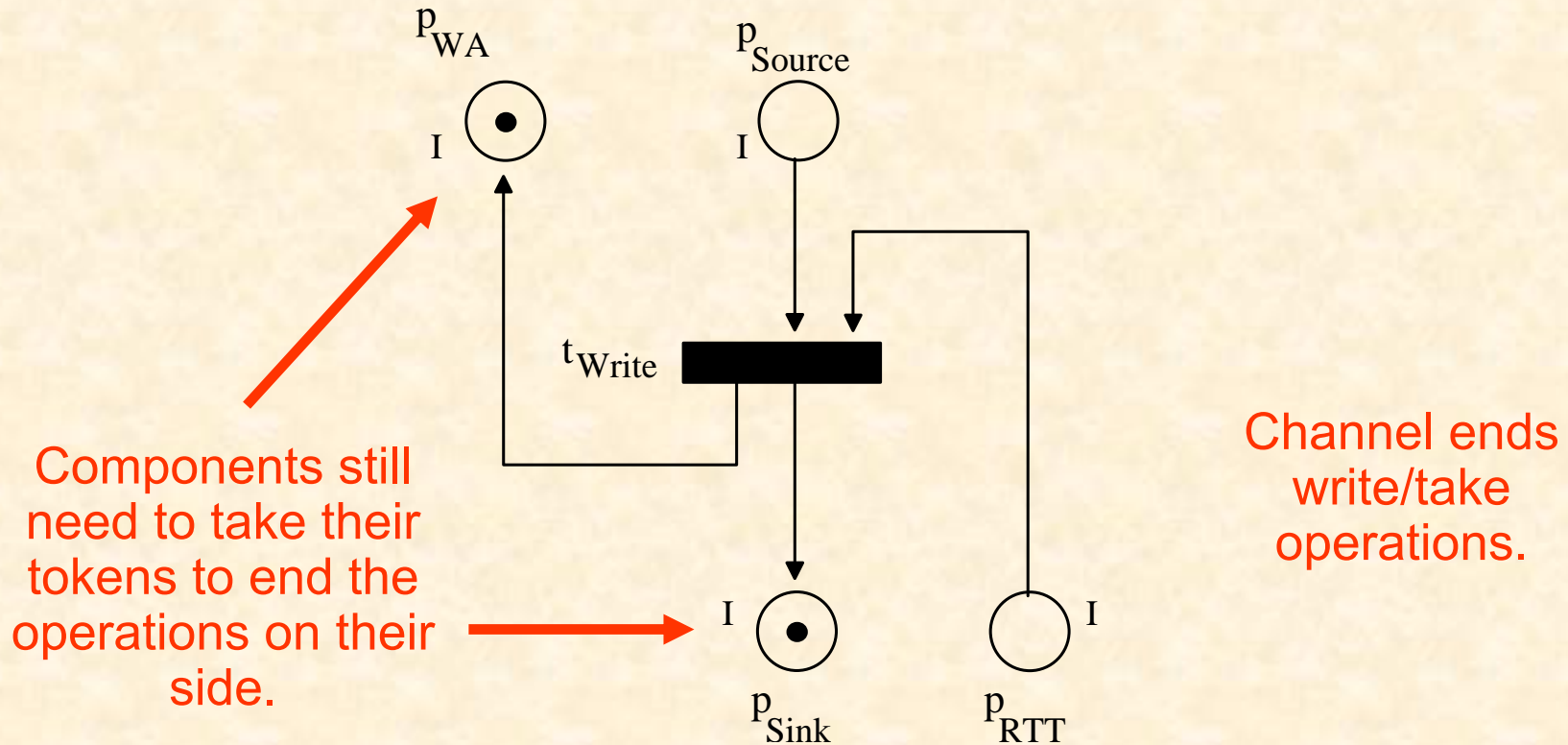
---



The channel is now ready to synchronize on the take and write operations.

A Component initiates a take operation.

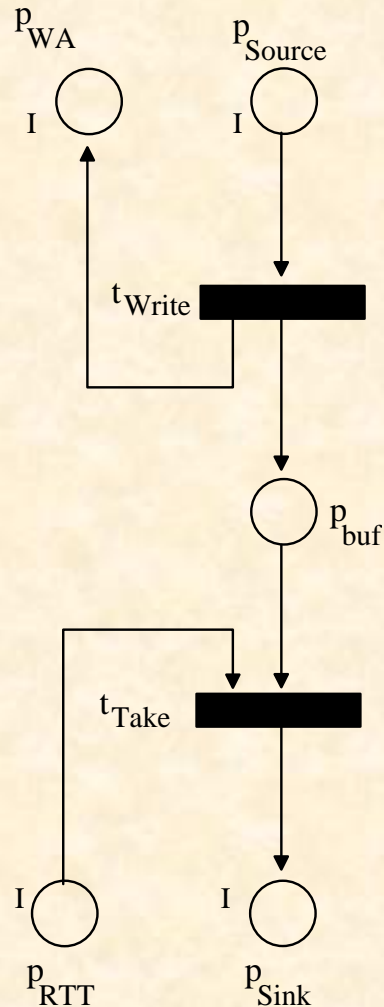
# Synchronous Channel(4)



$\{P_{source}, P_{rtt}\} [ T_{write} ] > \{P_{sink}, P_{wa}\}$

# FIFO-1 (1)

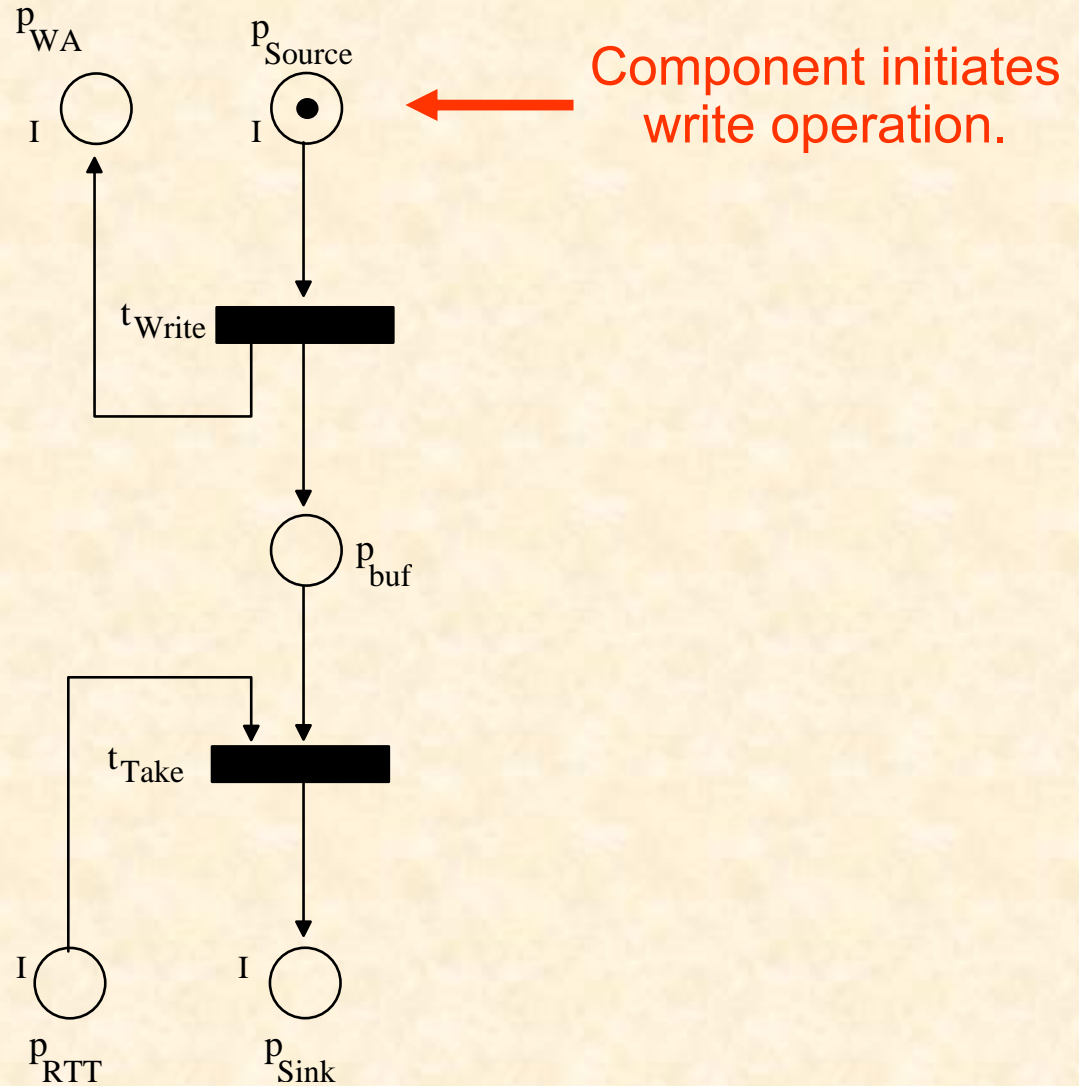
---



The I/O operations of both ends are done in an asynchronous way.  
However, there is an internal buffer of capacity one.

# FIFO-1 (2)

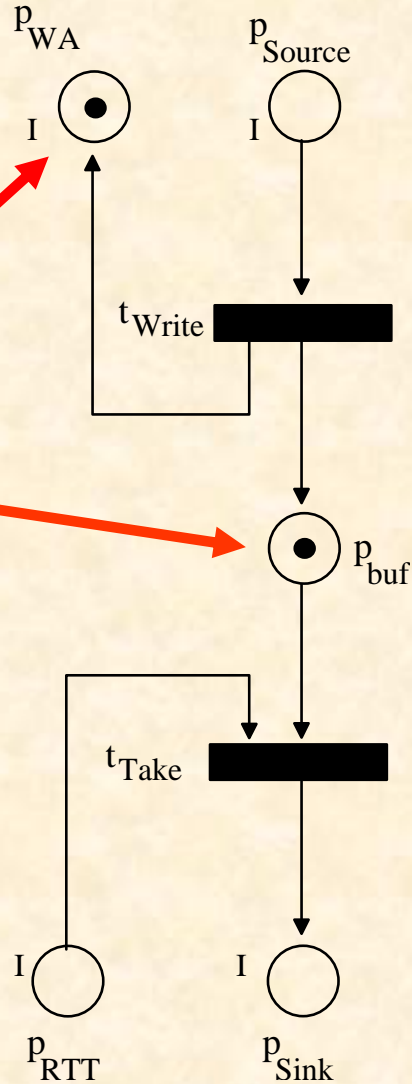
A first write:



# FIFO-1 (3)

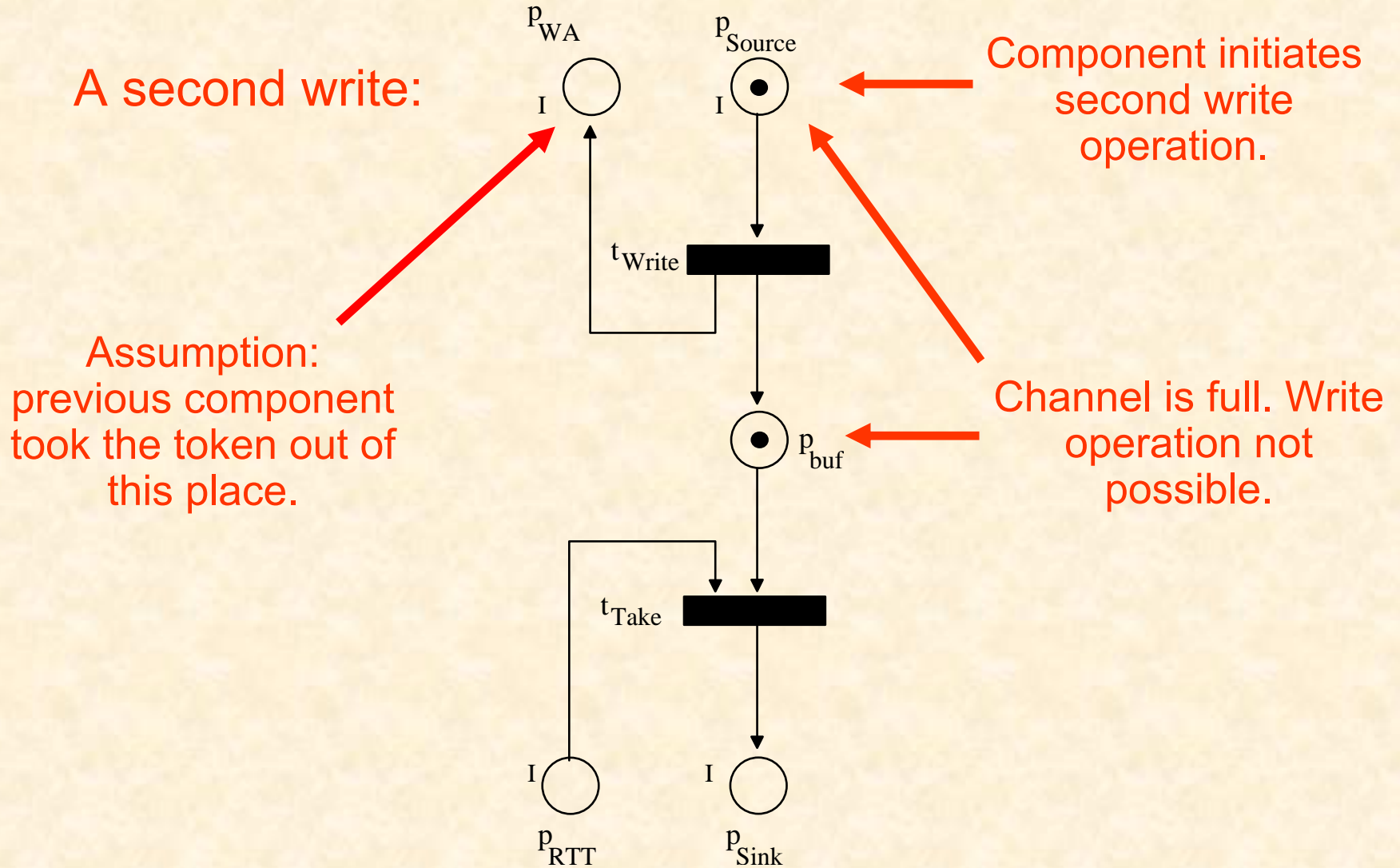
A first write:

Buffer was empty, so write succeeds.



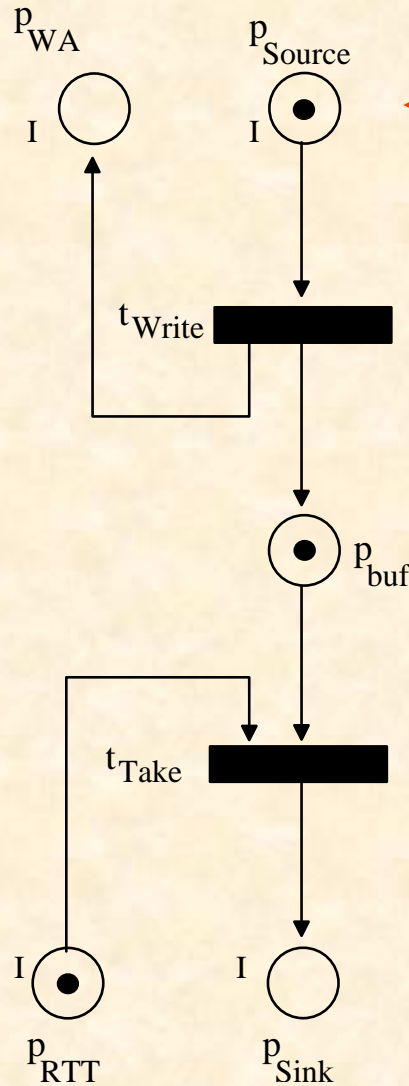
$\{P_{source}\} [ T_{write} ] > \{P_{buf}, P_{wa}\}$

# FIFO-1 (4)



# FIFO-1 (5)

A second write,  
and a take:

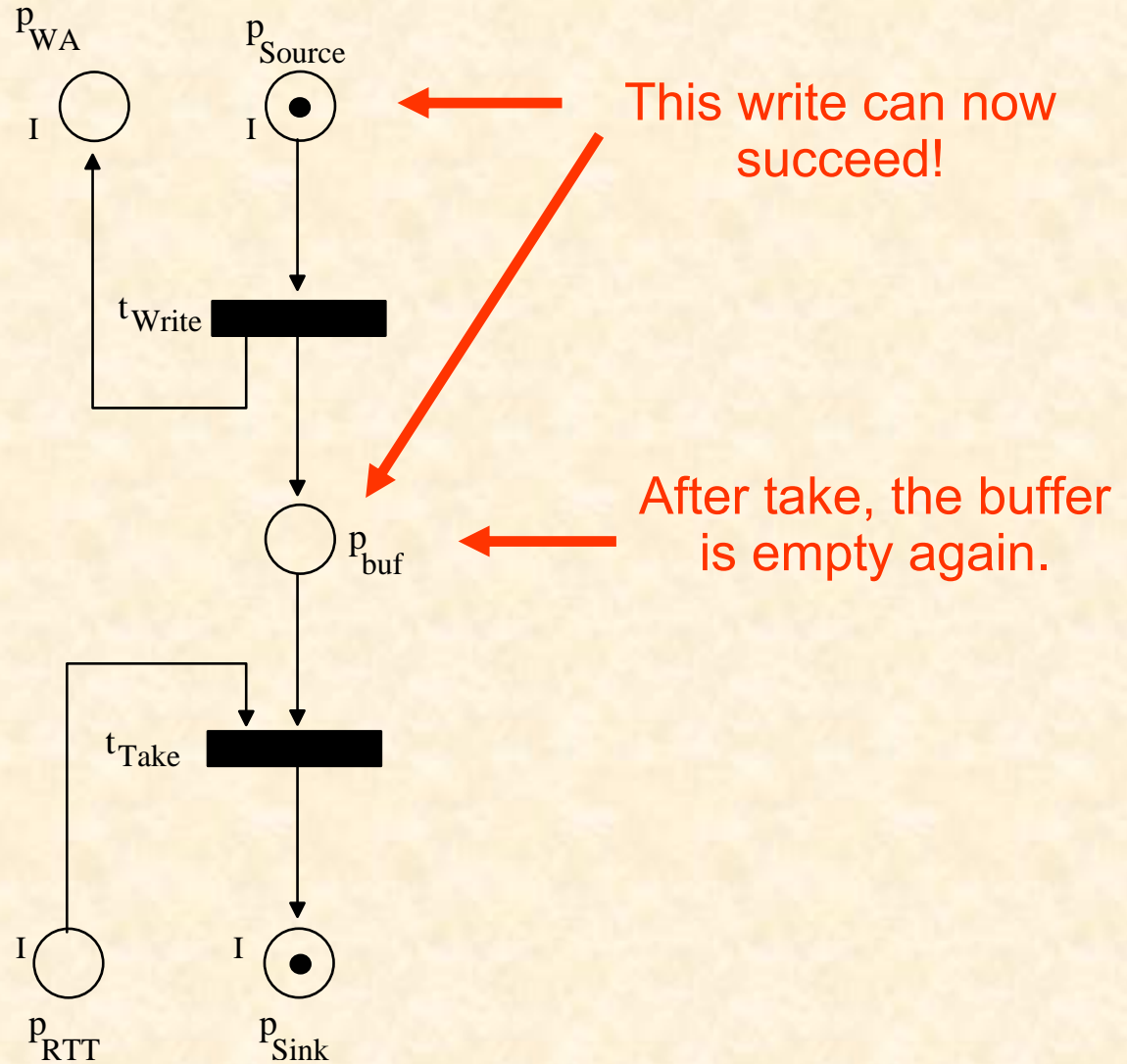


Still pending on write

Component initiates  
take operation.

# FIFO-1 (6)

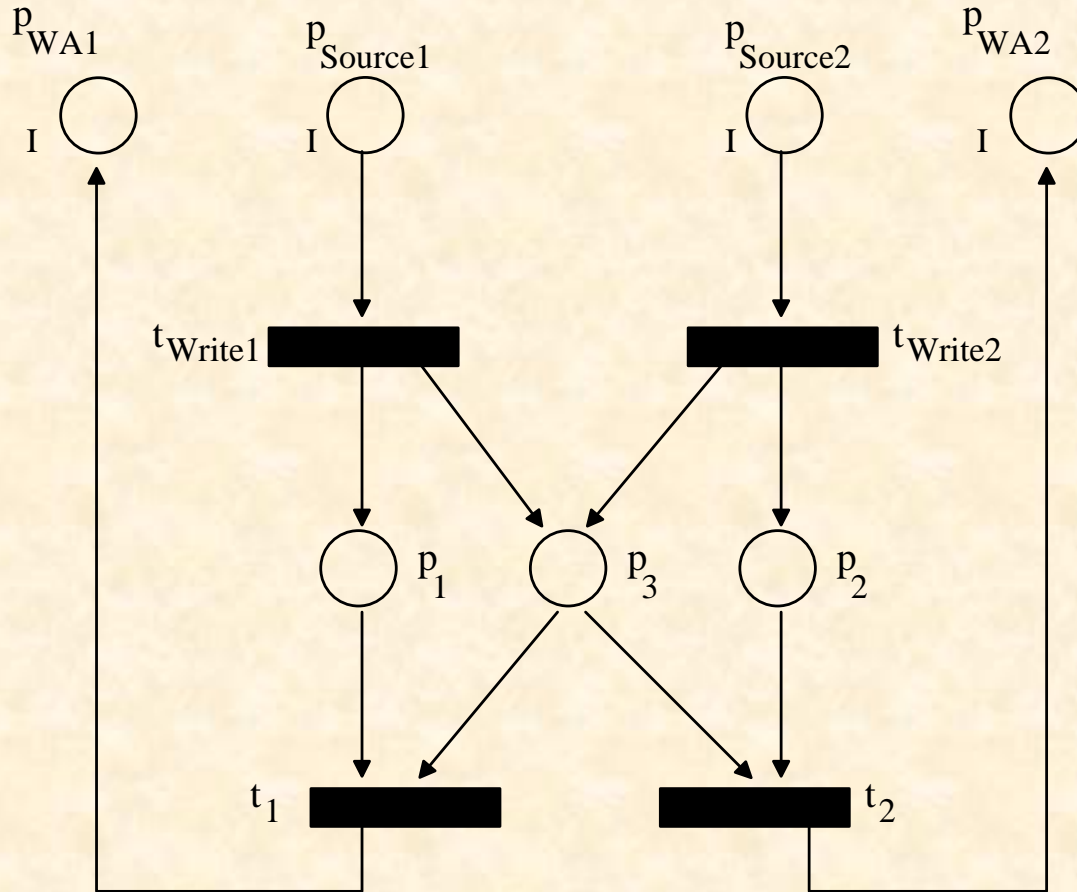
A second write.



$\{P_{buf}, P_{rtt}, P_{source}\} [ T_{take} > \{P_{sink}, P_{source}\}$

# Asynchronous Drain Channel(1)

---



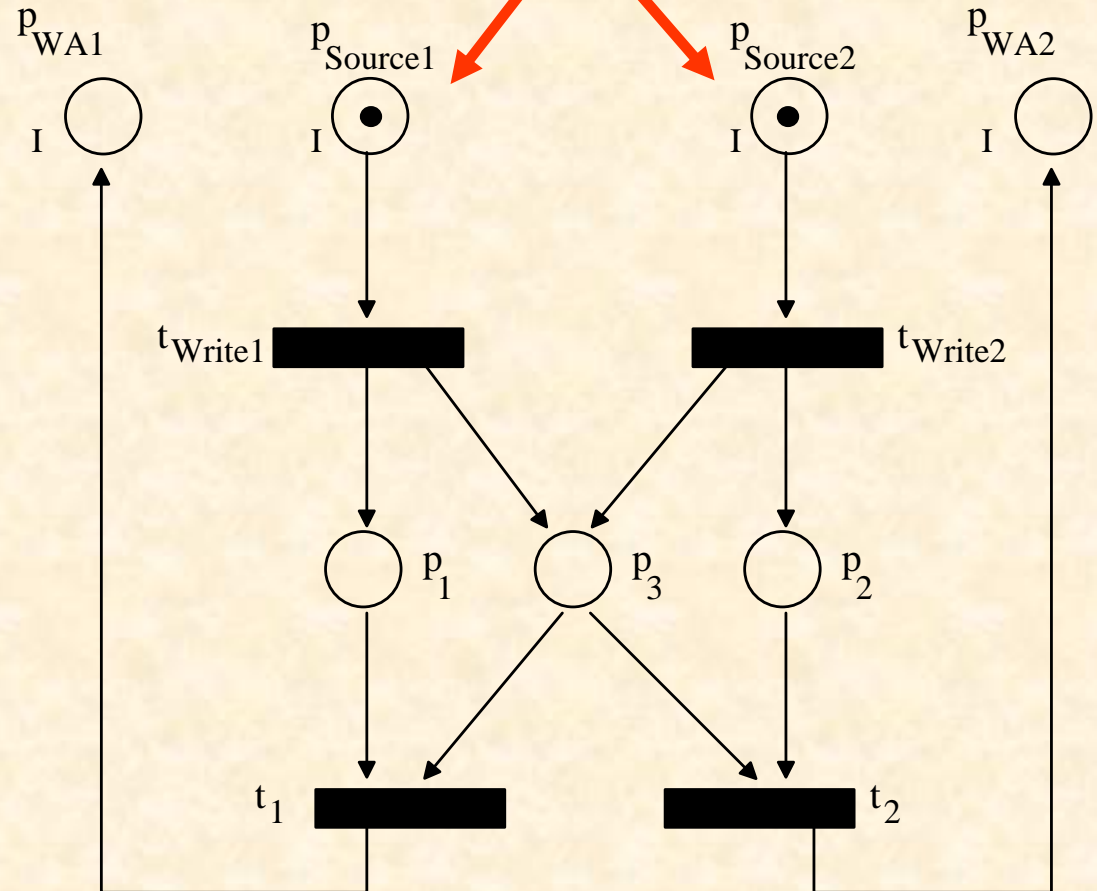
This channel type has two source-ends. The I/O operations performed on the ends succeed one at a time exclusively.

# Asynchronous Drain Channel(2)

Two simultaneous write attempts:

Only one write can succeed!

Non-deterministic choice on which write operation goes first.

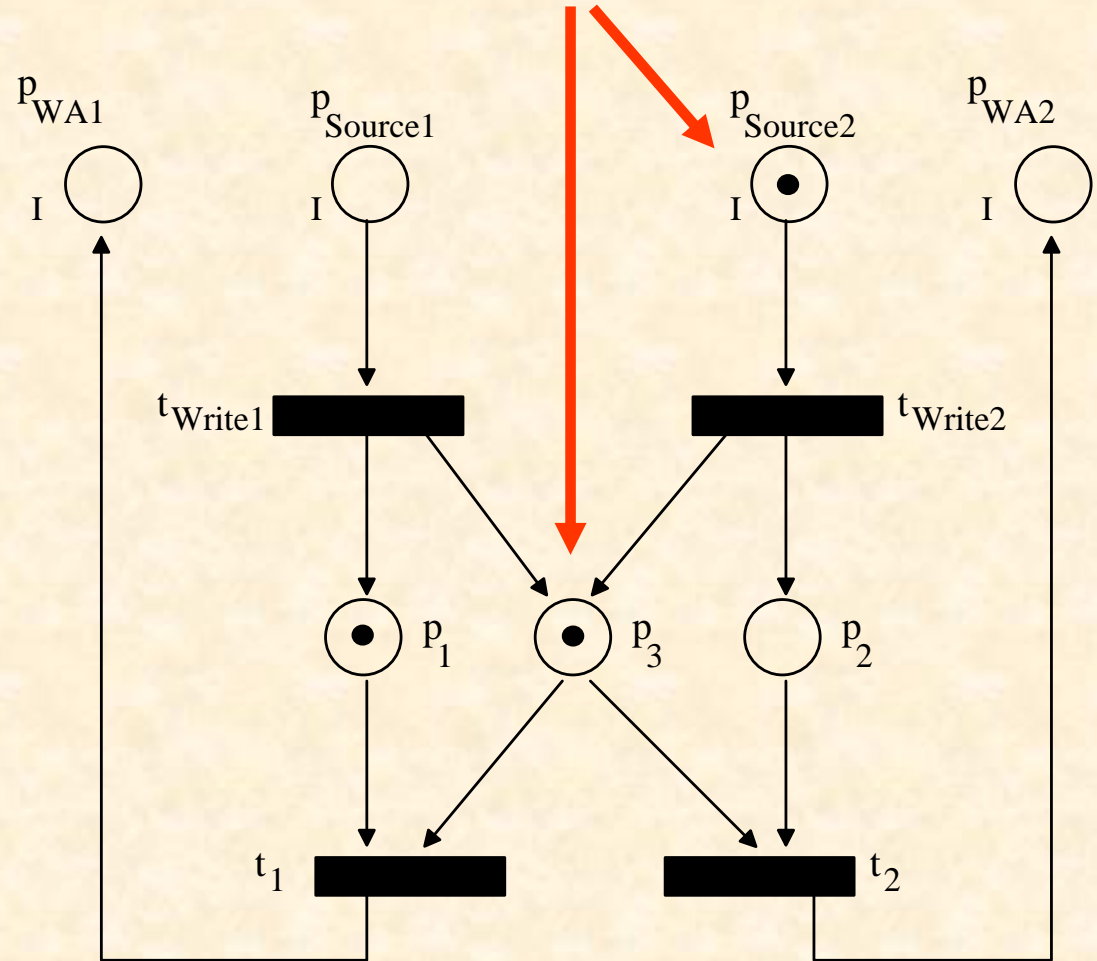


# Asynchronous Drain Channel(3)

Two simultaneous write attempts:

This write is blocked!

Example: we choose left write to succeed first.



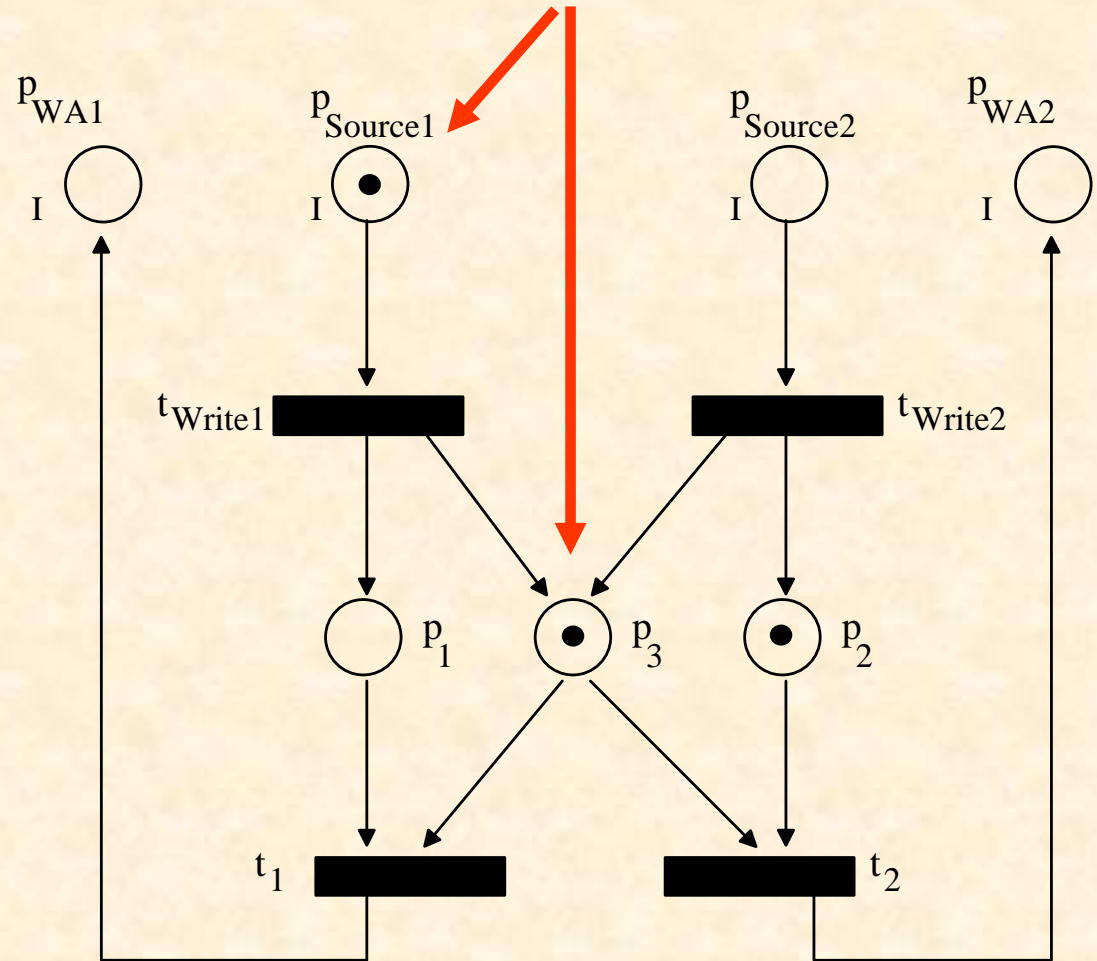
$\{P_{source1}, P_{source2}\} [ T_{write1} > \{P_1, P_3, P_{source2}\}$

# Asynchronous Drain Channel(4)

Two simultaneous write attempts:

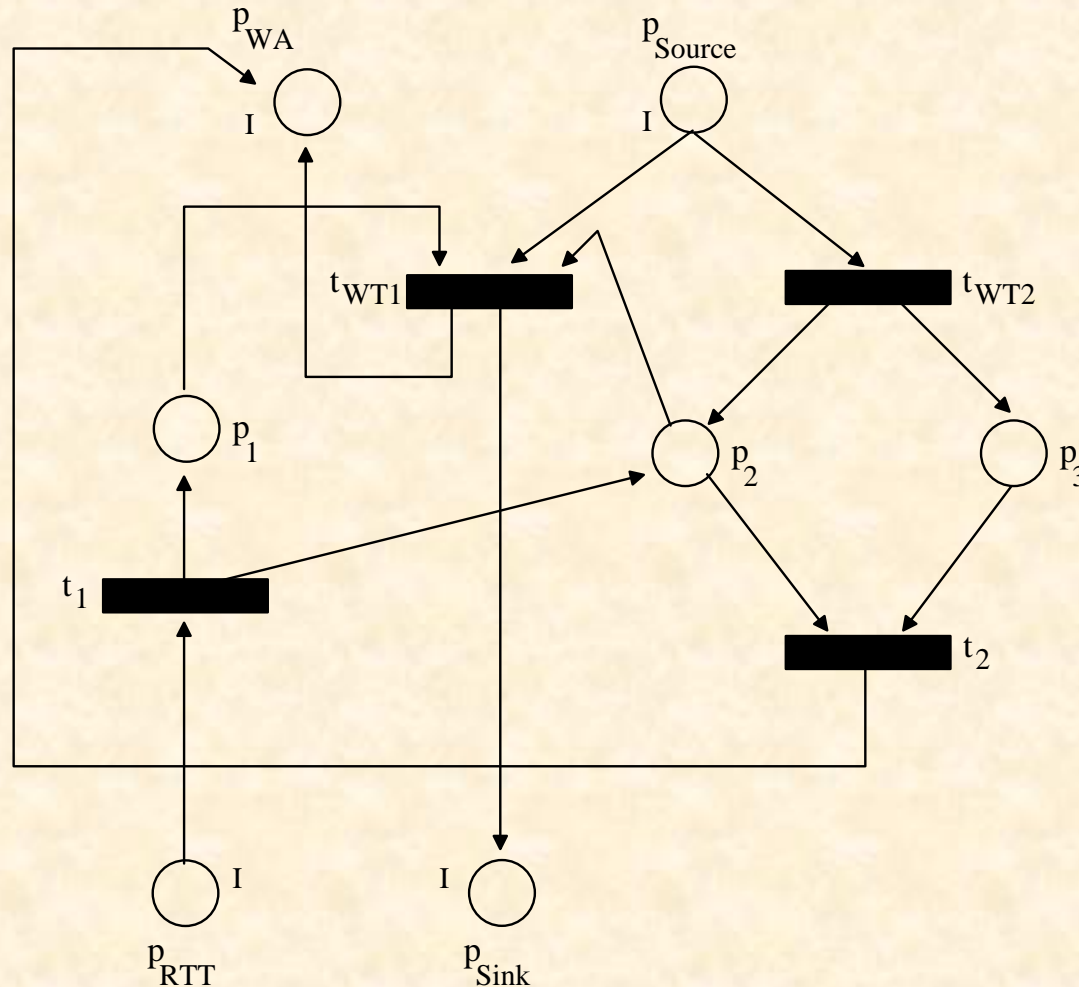
This write is blocked!

Example: we choose right write to succeed first.



$\{P_{source1}, P_{source2}\} [ T_{write2} > \{P_2, P_3, P_{source1}\}$

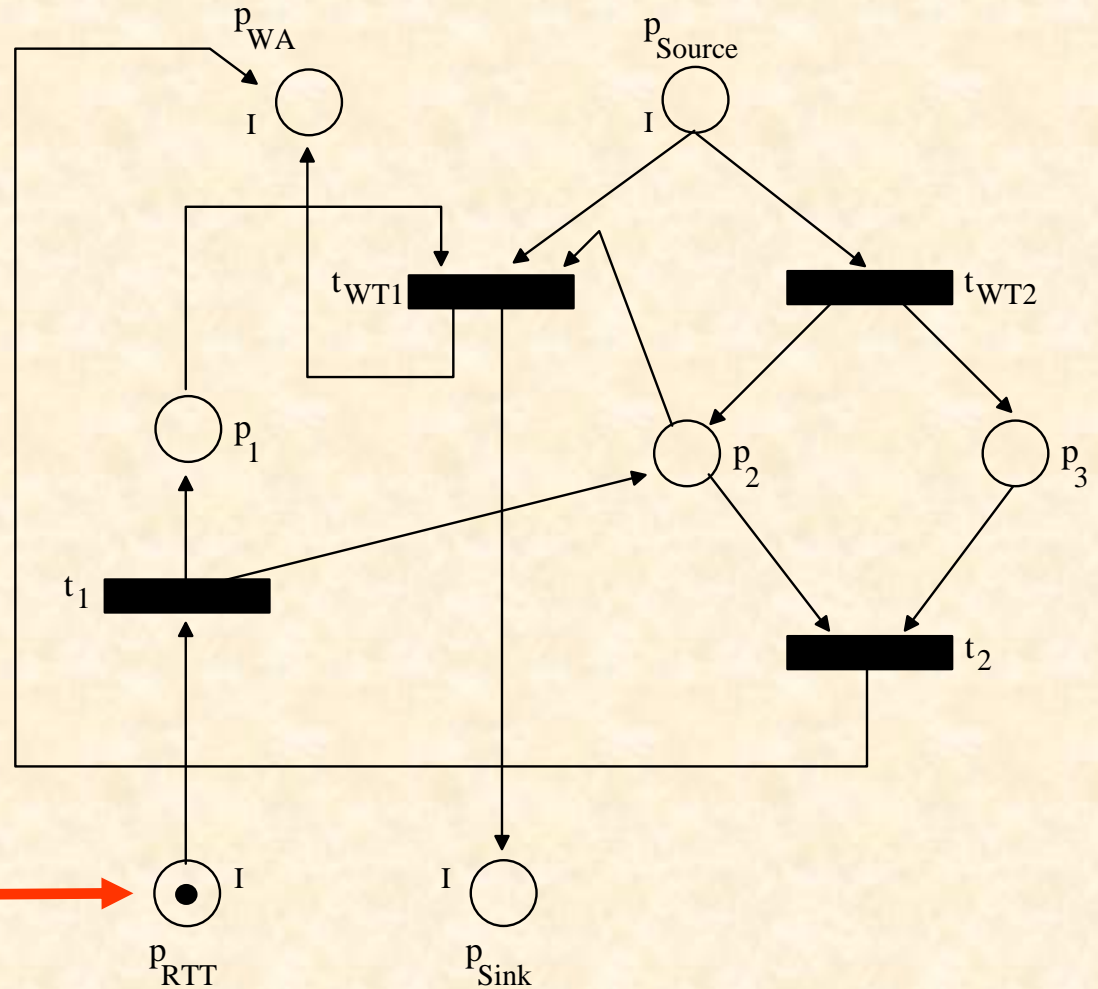
# Lossy Synchronous Channel(1)



If no take is being performed on the sink-end while writing a value to the source-end, the value gets lost. Otherwise, this channel behaves like a normal synchronous type.

# Lossy Synchronous Channel(2)

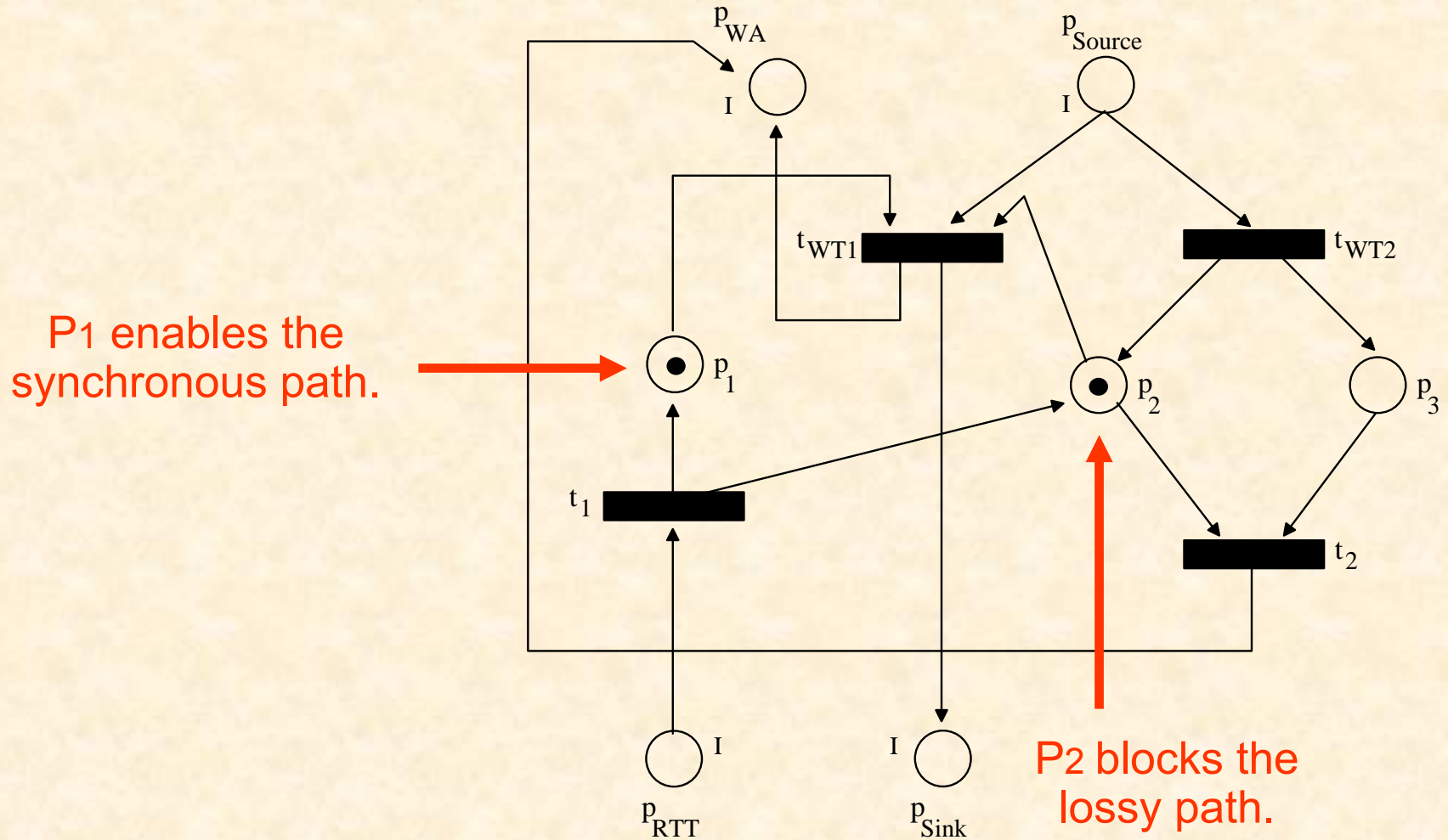
The synchronous path:



Component initiates  
take operation.

# Lossy Synchronous Channel(3)

The synchronous path:



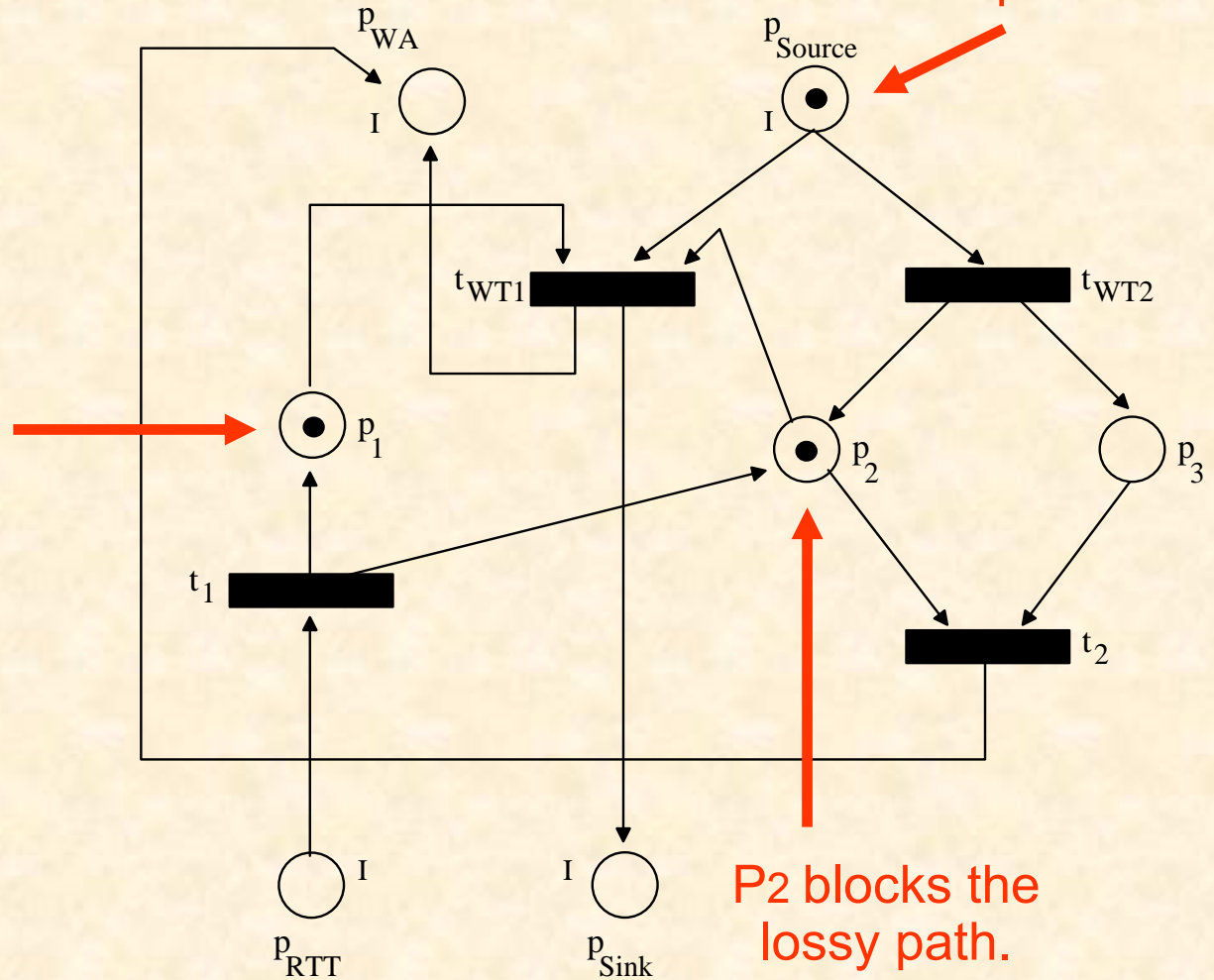
$\{P_{rtt}\} [ T1 > \{P1, P2\}$

# Lossy Synchronous Channel(4)

The synchronous path:

Component initiates write operation.

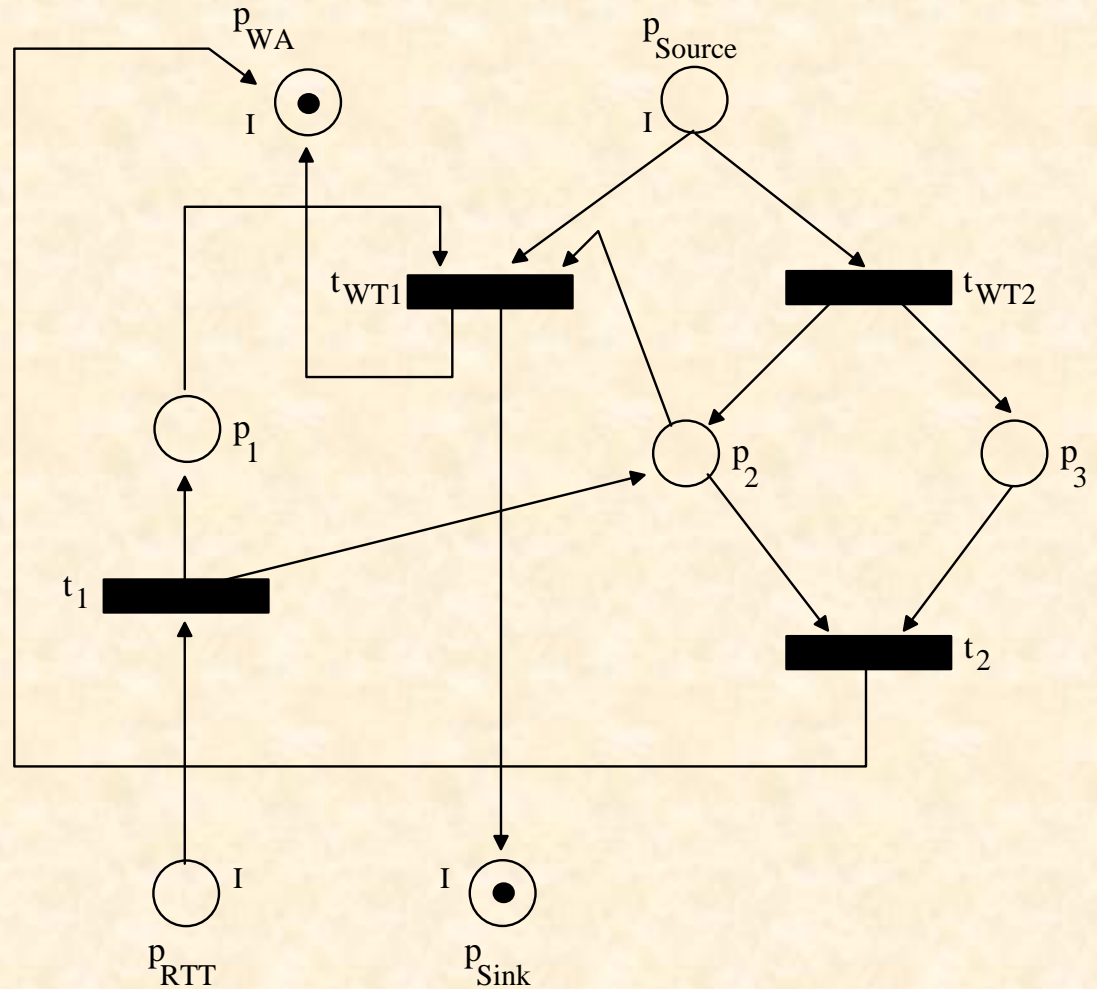
P1 enables the synchronous path.



# Lossy Synchronous Channel(5)

The synchronous path:

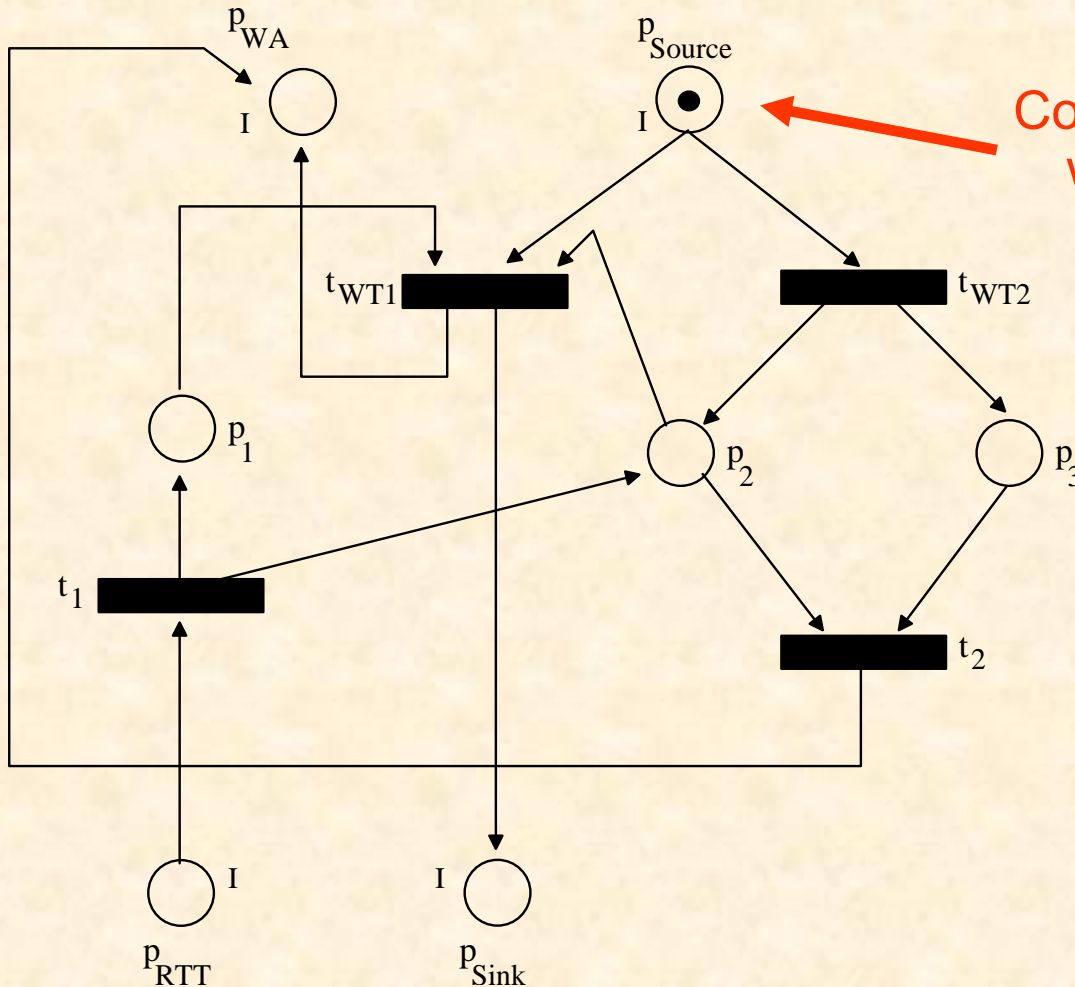
Channel  
synchronously  
performs  
write/take  
operation.



$\{P_1, P_2, P_{source}\} [T_{wt1}] > \{P_{sink}, P_{wa}\}$

# Lossy Synchronous Channel(6)

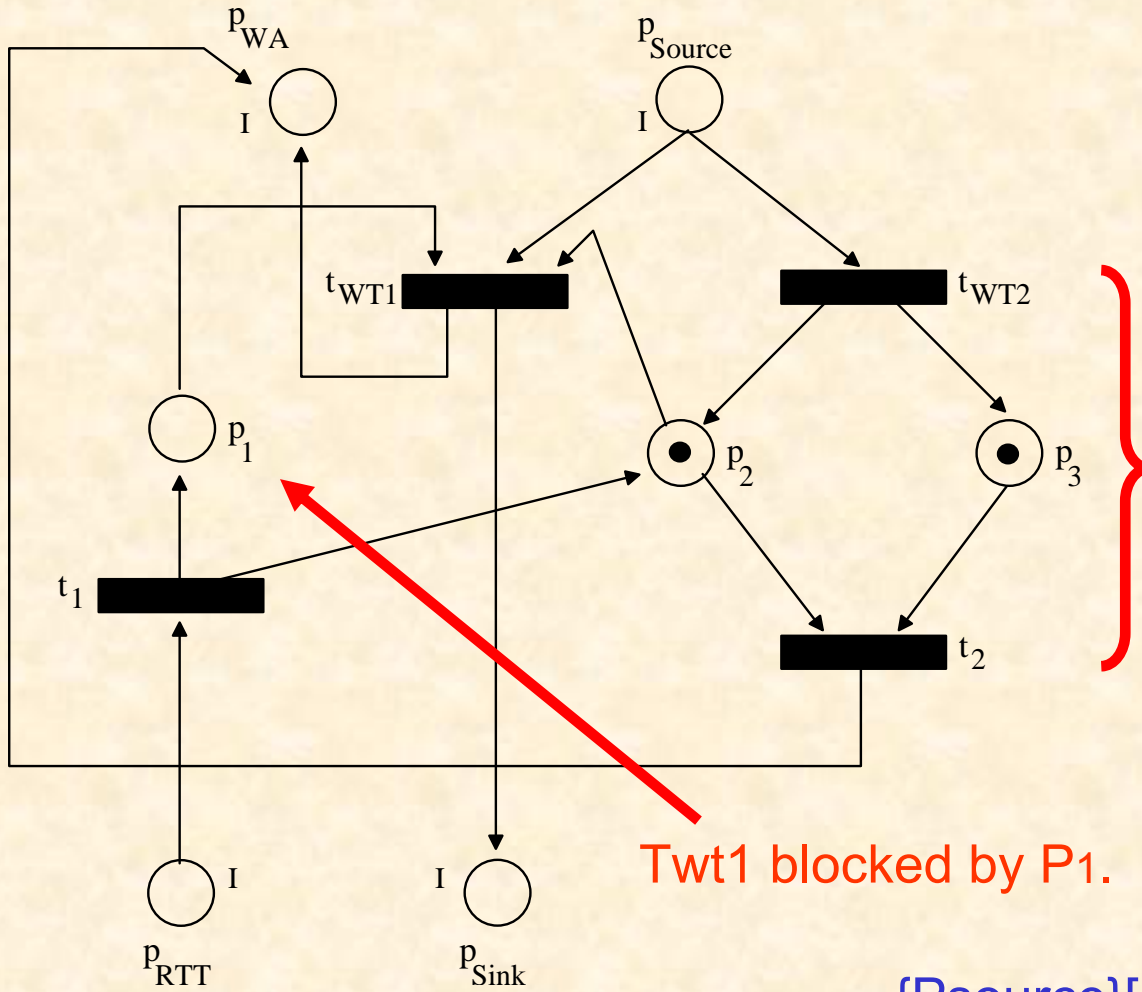
The lossy path:



Component initiates write operation.

# Lossy Synchronous Channel(7)

The lossy path:



No take, so we go with this path.

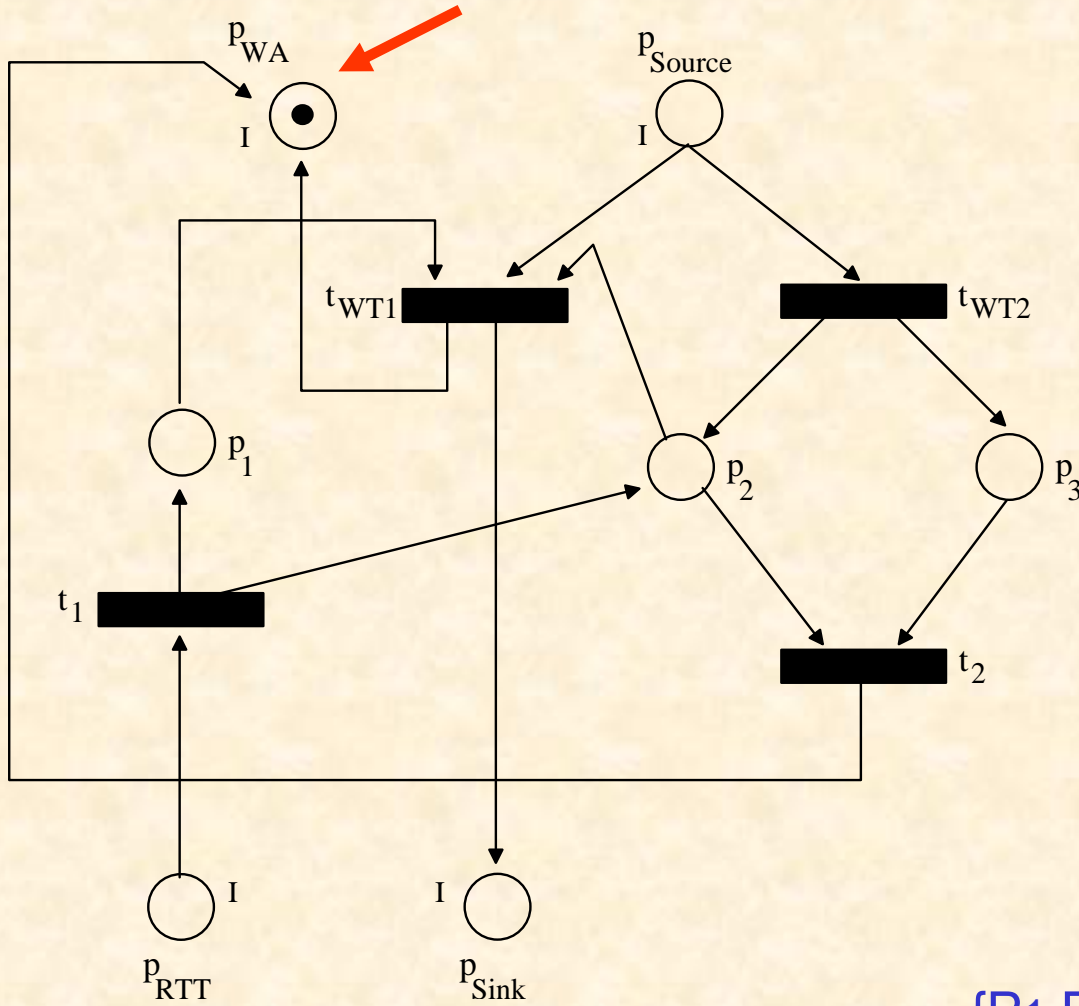
Twt1 blocked by P1.

$\{P_{Source}\} [T_{wt2} > \{P_2, P_3\}]$

# Lossy Synchronous Channel(8)

Write succeeds, value got lost.

The lossy path:



$\{P_1, P_3\} [ T_2 > \{P_{WA}\}$

# Analysis and Simulation

---

Simulation: by playing the token game

[see previous slides]

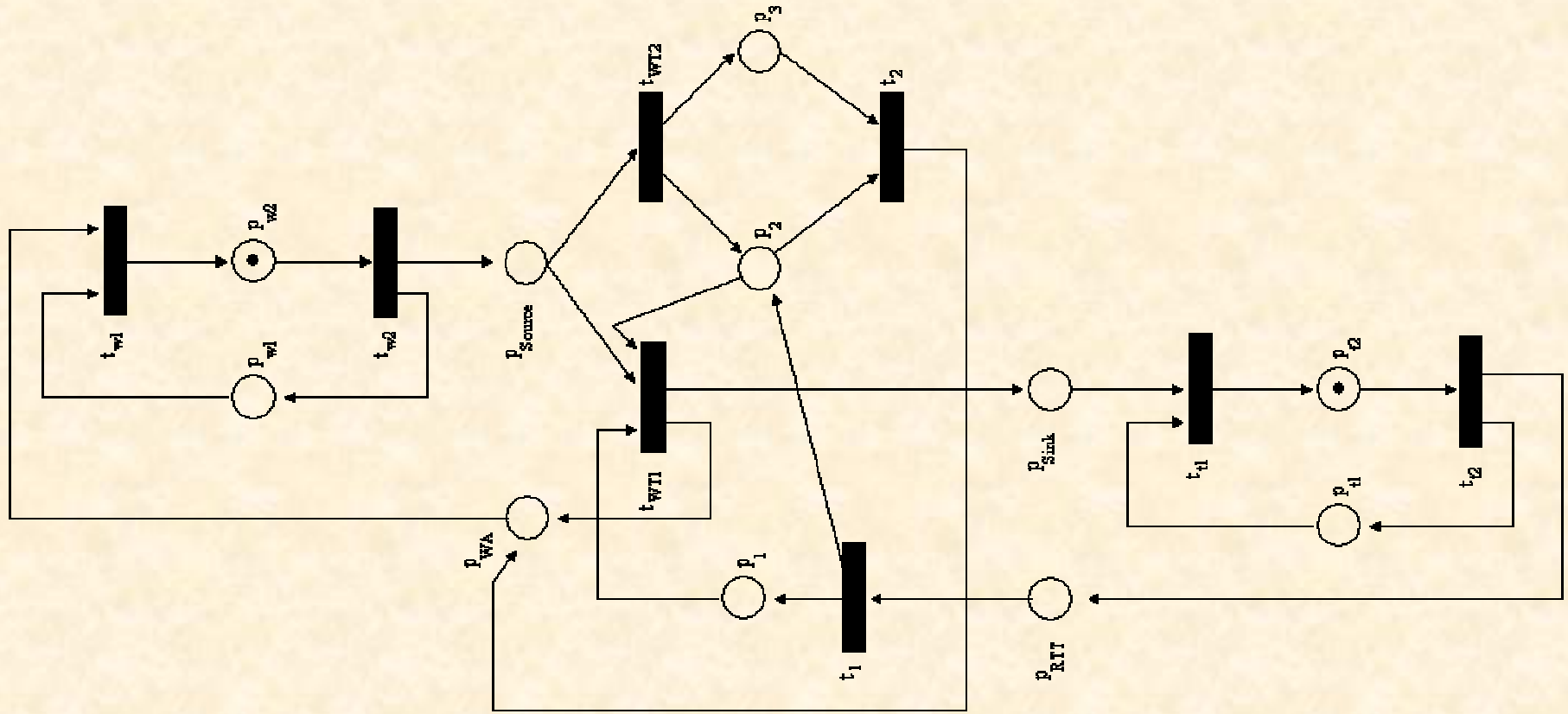
Analyze *exogenous coordination* behavior:

Most typical kind of analysis,

- ✓ Causality
- ✓ Concurrency
- ✓ Conflicts
- ✓ Confusions
- ✓ Deadlocks
- ✓ Equivalence

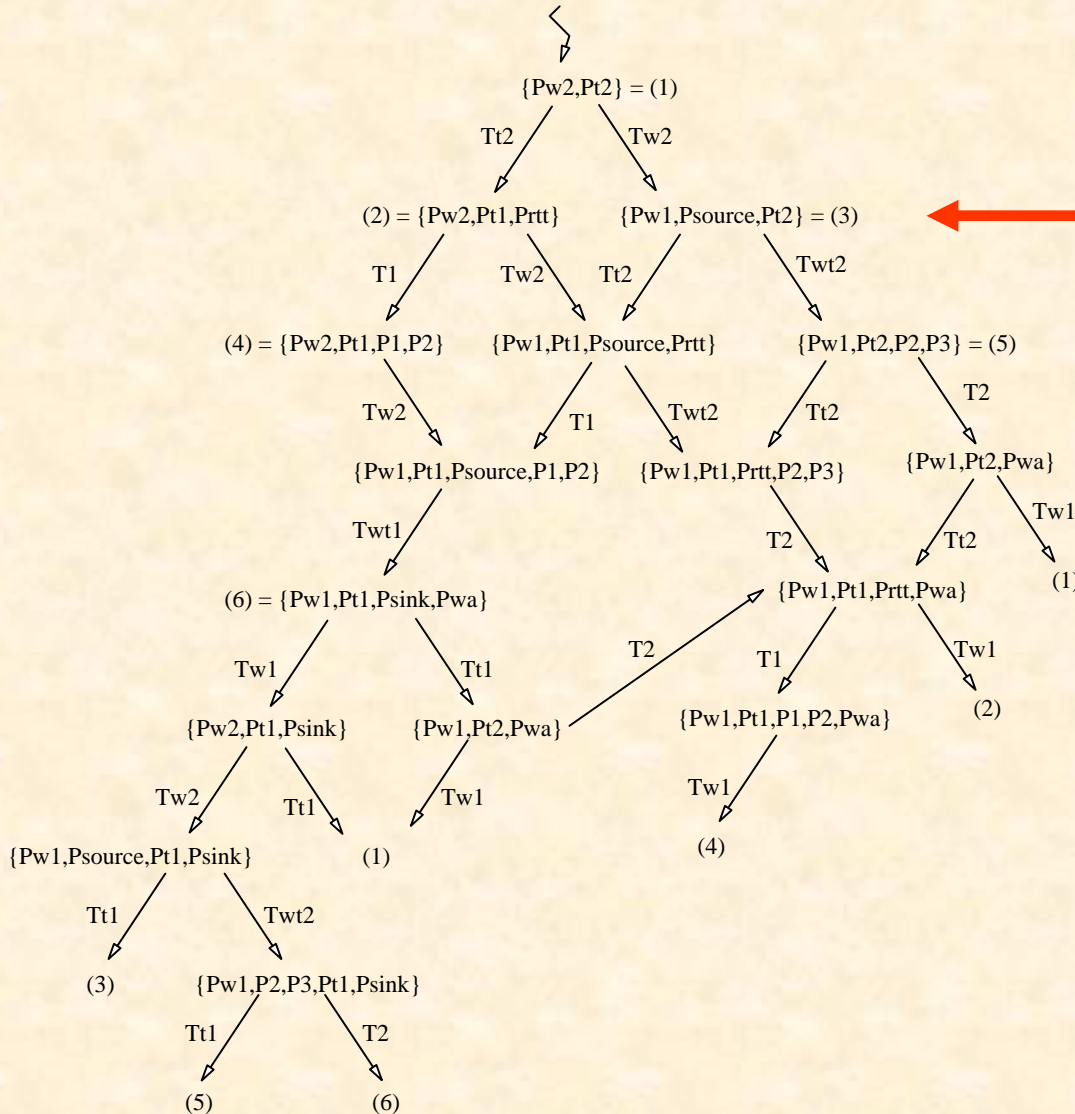
# Complexity (1)

Example,  
a writer + lossy synchronous channel + a taker:



# Complexity (2)

A graph with all possible configurations:



All previous analysis can be deduced from this graph.

However, it is already not a pleasant task to simulate and analyze "by hand".

# Complexity (3)

---

A real application consists of many components and channels.

The negative side of our approach:

The PN model gets really big and is not tractable for humans anymore.

We recommend using,  
higher level Petri Nets, such as Colored Petri Nets,  
the MoCha-pi calculus,  
Reo,  
if human readability is desired.

# Complexity (4)

---

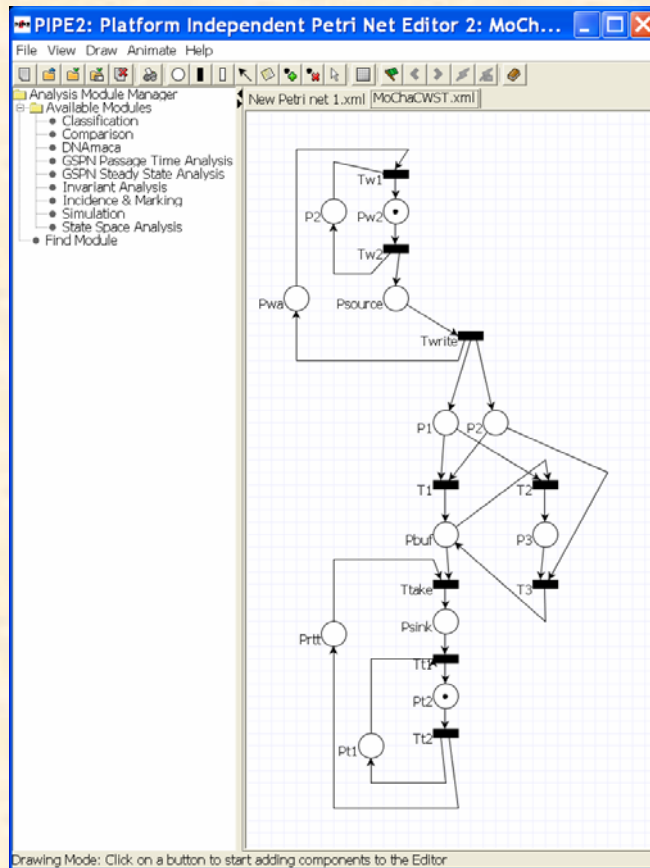
The positive side of our approach:

Our models are very suitable for analysis and simulation using tools.

- + P/T and EN systems have clear non-changeable semantics rules and constructs = one level of abstraction. [unlike higher PN levels]
- + we model technical details that are otherwise omitted in higher levels.
- + there are many tools available for this kind of PN's.

# Tool Example

The *Platform Independent Petri Net Editor* tool (PIPE).



State Space Analysis

Source net

Use current net Filename: pipeBeta15MoChCWST.xml Browse

Results

**Petri net state space analysis results**

Bounded: false

Safe: false

Deadlock: false

Copy Save

Analyse

Simulation

Source net

Use current net Filename: pipeBeta15MoChCW... Browse

Simulation parameters

Firings: 1100 Cycles: 5

Results

**Petri net simulation results**

Place	Average number of tokens	95% confidence interval (+/-)
Pw2	0.30893	0.0621
Psource	0.23762	0.10414
Pt1	0.74257	0.14175
Ptt	0.53465	0.19273
P2	0.69307	0.0621
Pwa	0.45545	0.09152
P1	0.34653	0.58261
P2	0.39604	0.67093
Pbuf	0.50495	0.52465
P3	0.0495	0.10629
Psink	0.20792	0.07198
Pt2	0.25743	0.18175

Copy Save

Simulate

# Summary & Conclusions(1)

---

- ✓ We model the exogenous coordination behavior of distributed mobile channel based systems using Petri Nets.
- ✓ We get for free all the extensive theoretical work already done for PNs.
- ✓ We can use one of the many tools available for PNs.
- ✓ We defined a method to construct PN models from components and channels that is compositional.
- ✓ For this we gave a composition function on (interface) places.
- ✓ We gave PN's for the minimal behaviour that components need to exhibit towards channels.

# Summary & Conclusions(2)

---

- ✓ We gave PNs for a set of representative channel types.
- ✓ We discussed analysis and simulation of our models.
  - Token game,
  - causality,
  - concurrency,
  - conflicts,
  - confusions,
  - deadlocks, and
  - equivalence.

# Summary & Conclusions(3)

---

- ✓ We discussed the complexity of our approach:
  - On the negative side, the models that we produce quickly become intractable for humans.
  - + On the positive side, the low-level semantics of these models makes them very suitable for simulation and analysis using one of the many tools available for PNs.
- ✓ We gave an example of such a tool.