Petri Net Patterns

Lecture 71, v01

John G. Artus

BSEE

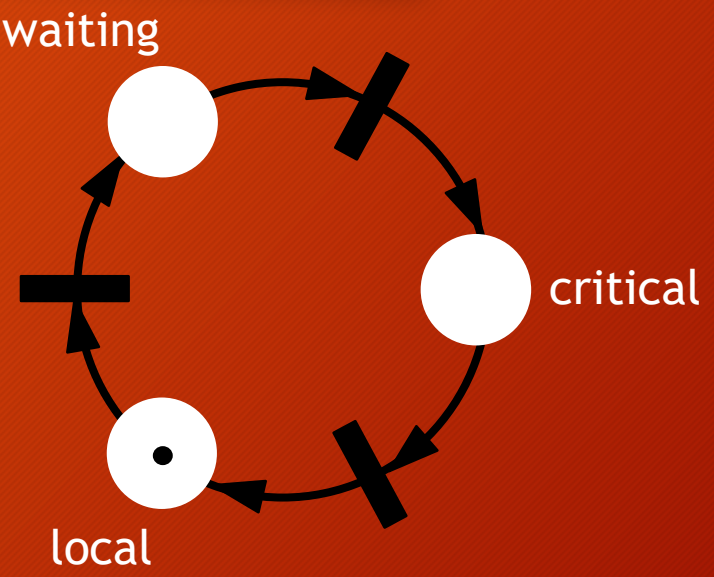
MSSE

INCOSE ESEP

Mutual Exclusion

Process Stages

- Often, when representing a system process, we show the process having three stages of operation
- Local
  - The system is performing some internal actions in preparation for performing the critical action
- Waiting
  - The internal actions have completed, and the system is now waiting in a queue to perform the critical action
- Critical
  - The system is performing the critical action
  - When this action is complete, the token is passed to “local” indicating that another process can begin



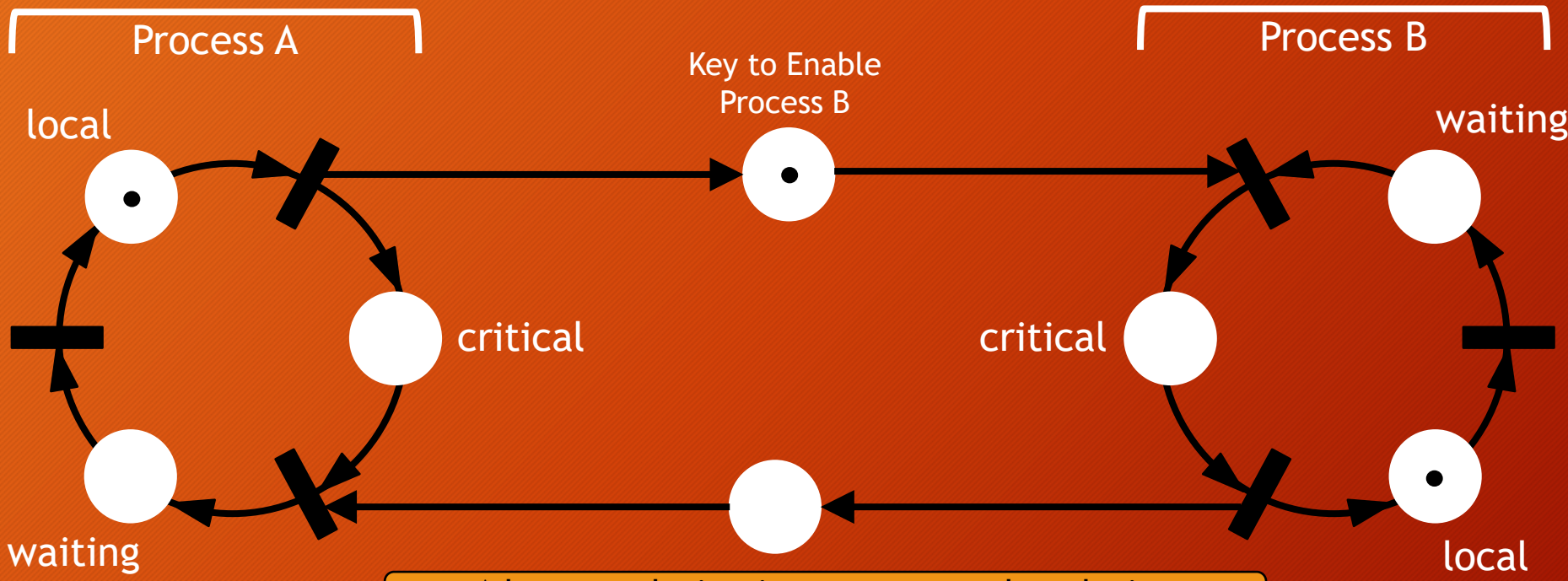
It is not uncommon to have system processes that can perform multiple operations (jobs) simultaneously

This would be represented by having multiple tokens, each one indicating a separate operation

Thus, multiple critical actions can be performed at the same time

Coordinating Multiple Processes

- A common way of coordinating multiple processes is to use a control key that is shared between the processes such that only one process can operate at a time
  - This assumes that both processes access critical resources that cannot be shared
  - And therefore the need to have only one process operate at a time
- One way to handle this is to use the control key to alternate between two processes



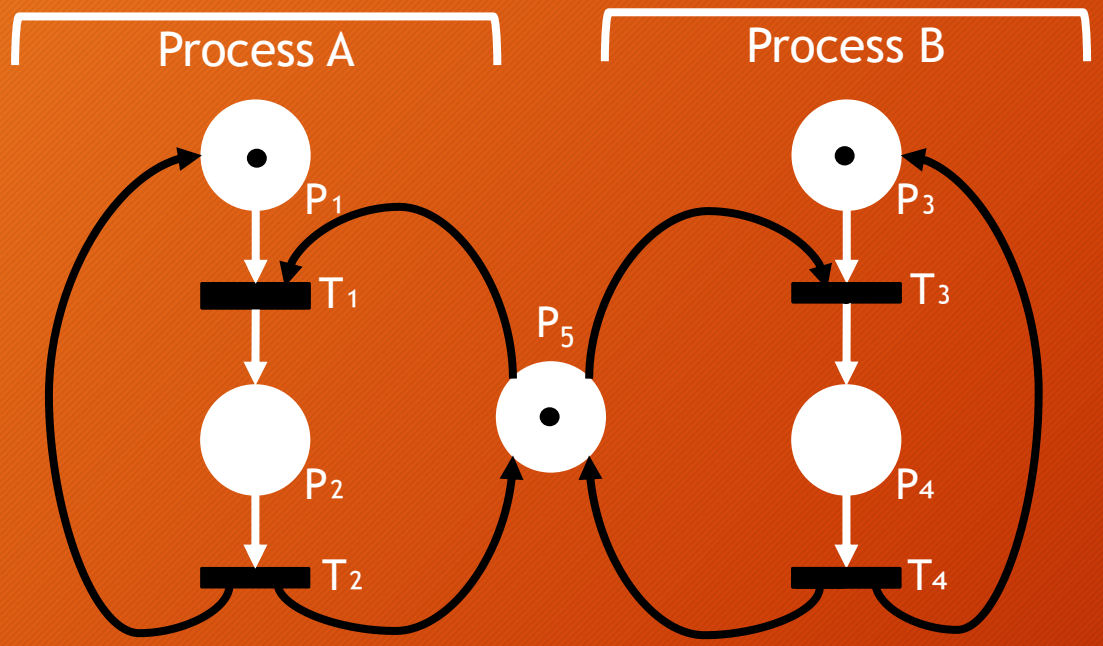
However, in this arrangement, if one process is stuck in the local stage performing some internal operation, then it can dominate the entire mechanism and prevent the other process from performing its critical operation

This occurs because the key is dedicated to that process and is waiting for the job to move to the waiting stage

A better solution is to use mutual exclusion

Mutual Exclusion

- Mutual exclusion occurs when two processes are coordinated by a central controller such that only one process operates at a time



Place P5 (the central controller) is a "non-deterministic" node

Meaning that when simulated, it will issue its token randomly

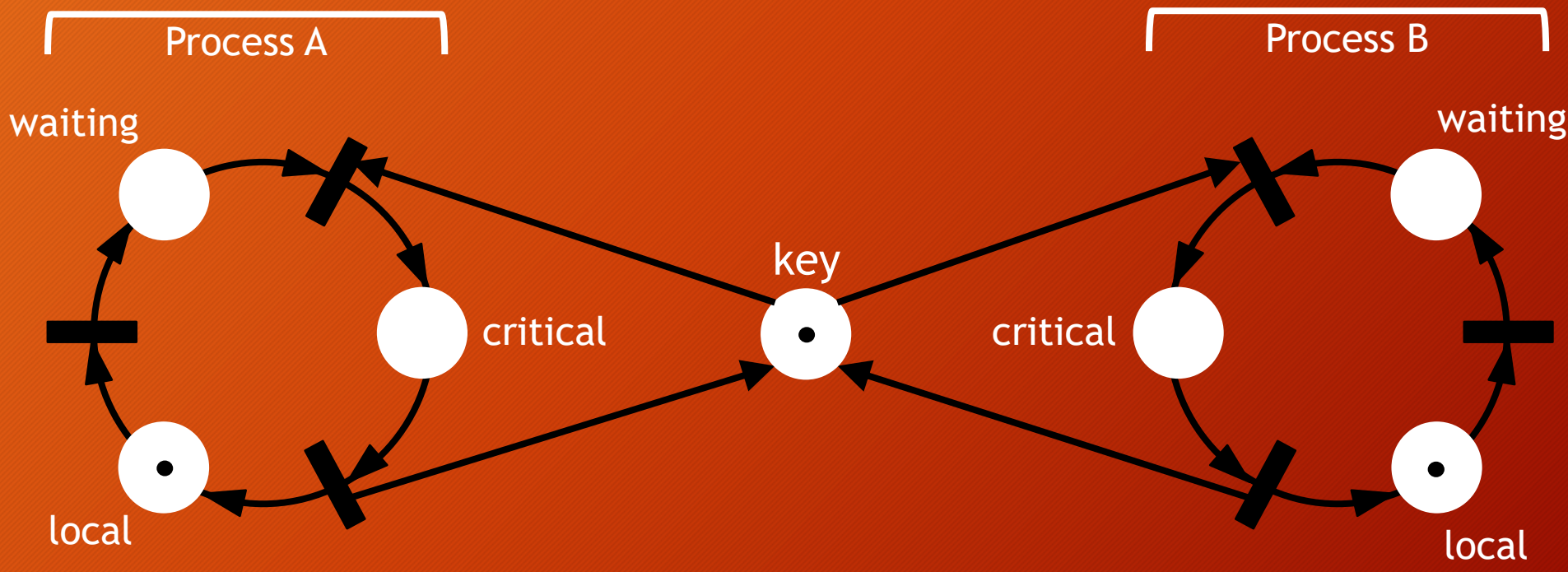
The critical state for Process A is P2

The critical state for Process B is P4

- In this example, two processes are operating concurrently
  - Each has a token that is ready to be supplied to the process
  - But the first transition needs the control "key" token as well in order to fire
  - The central controller can only issue the key to Process A or Process B, not to both
  - So, effectively, only one process can operate at a time
  - When the process completes, the key is returned to the central controller

Coordinating Processes with Mutual Exclusion

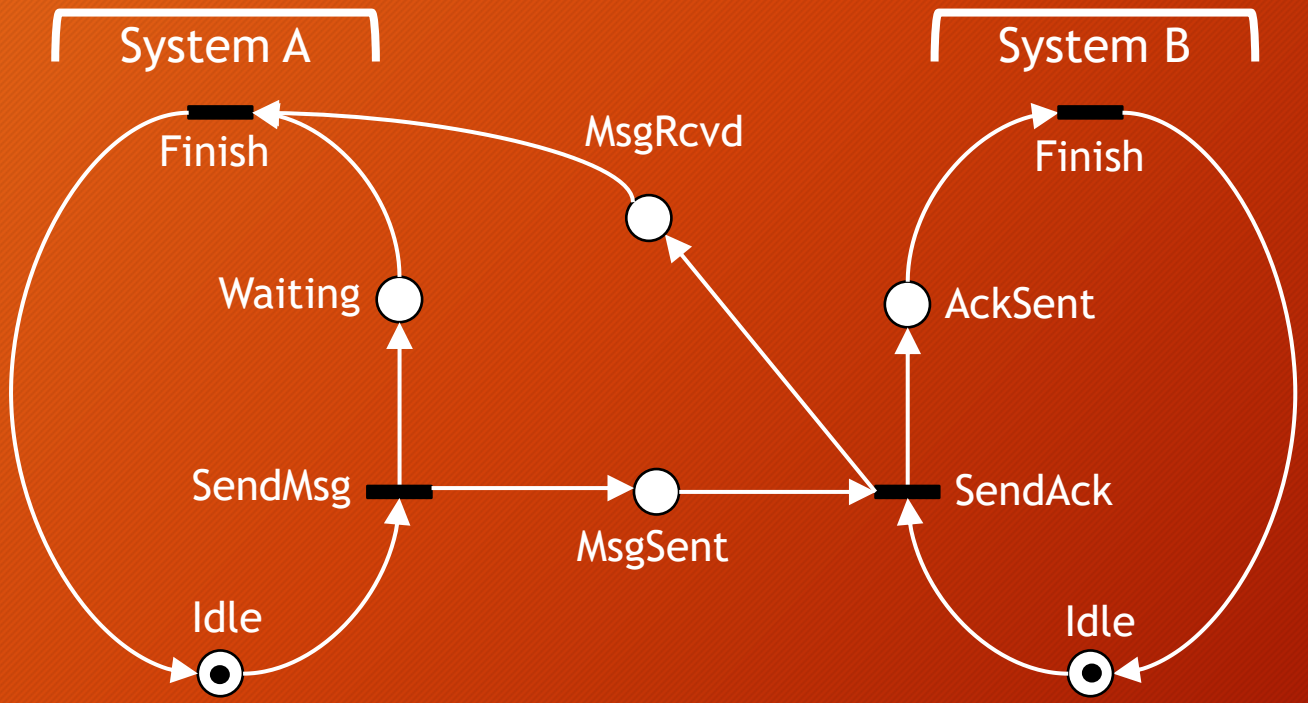
- Mutual exclusion is a common way of coordinating two or more processes
  - As in the alternating control of processes, only one process operates at a time
  - However, if one process is stuck performing some local action, the other process can obtain the key to keep working
  - This occurs because the key is available to both processes
  - The key will go to the process that first has a job that reaches the waiting stage



Avoiding Crosstalk

Two Systems in One-Way Communication

- In communication, two systems can coordinate with each other using a message / acknowledge scheme
- This allows one system (System A) to send messages to (System B) with high assurance that System B will receive the message
  - This is because System A waits for an acknowledgement received from System B before sending the next message
  - This avoids sending messages while the receiving system is not prepared to accept the new incoming message
- The acknowledgement is an indication that System B is ready to receive the next message



The scheme depicted here works because the two transmissions flow at separate times, even if on the same transmission channel

But the shceme only works for one-way transmission of messages

Two Systems in Two-Way Communication

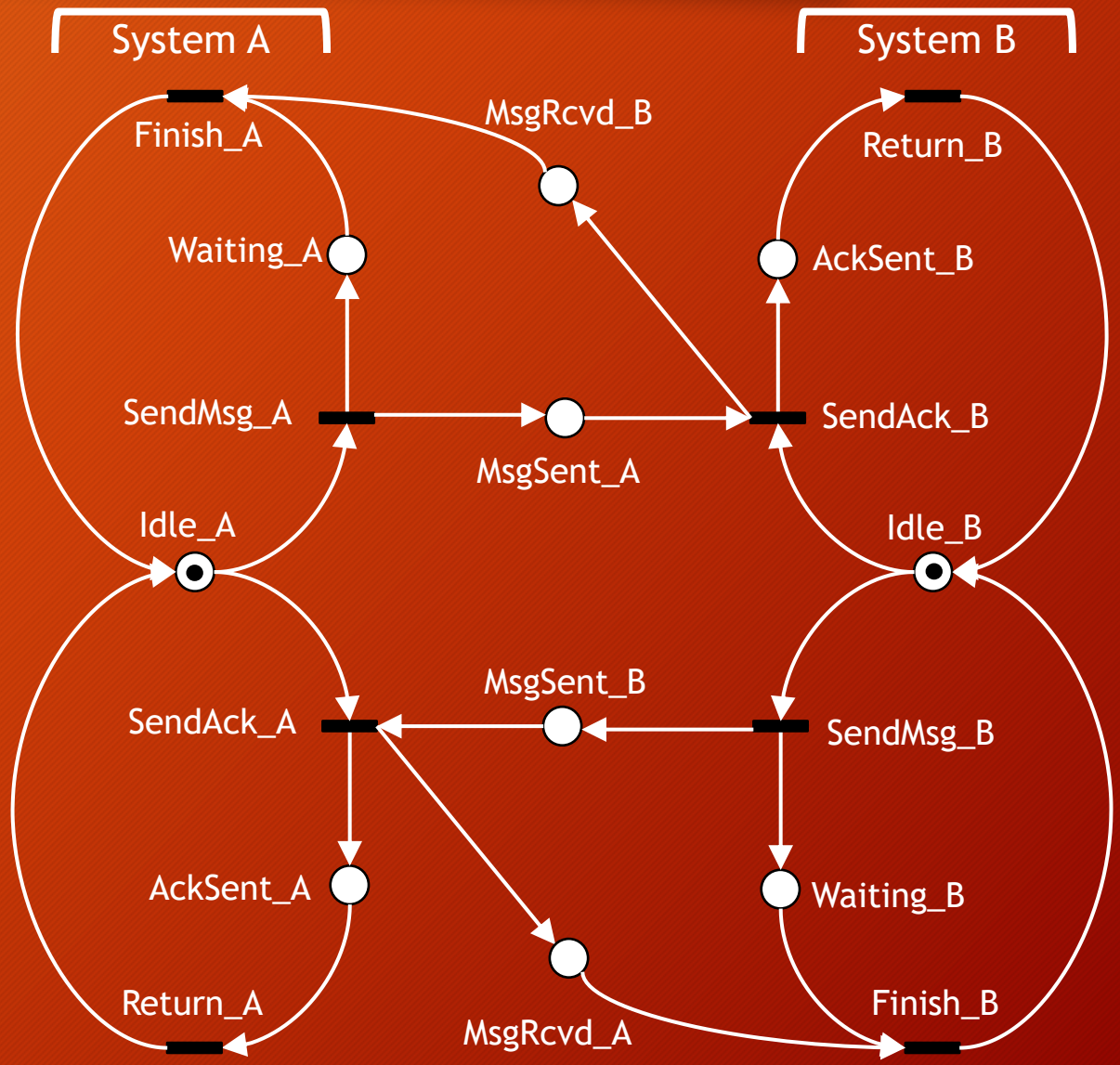
- One might think that doubling this arrangement might allow two-way communications

Since both systems start by sending their first message simultaneously, there is no token in the Idle state to use to trigger and acknowledgement

The two systems become “deadlocked”

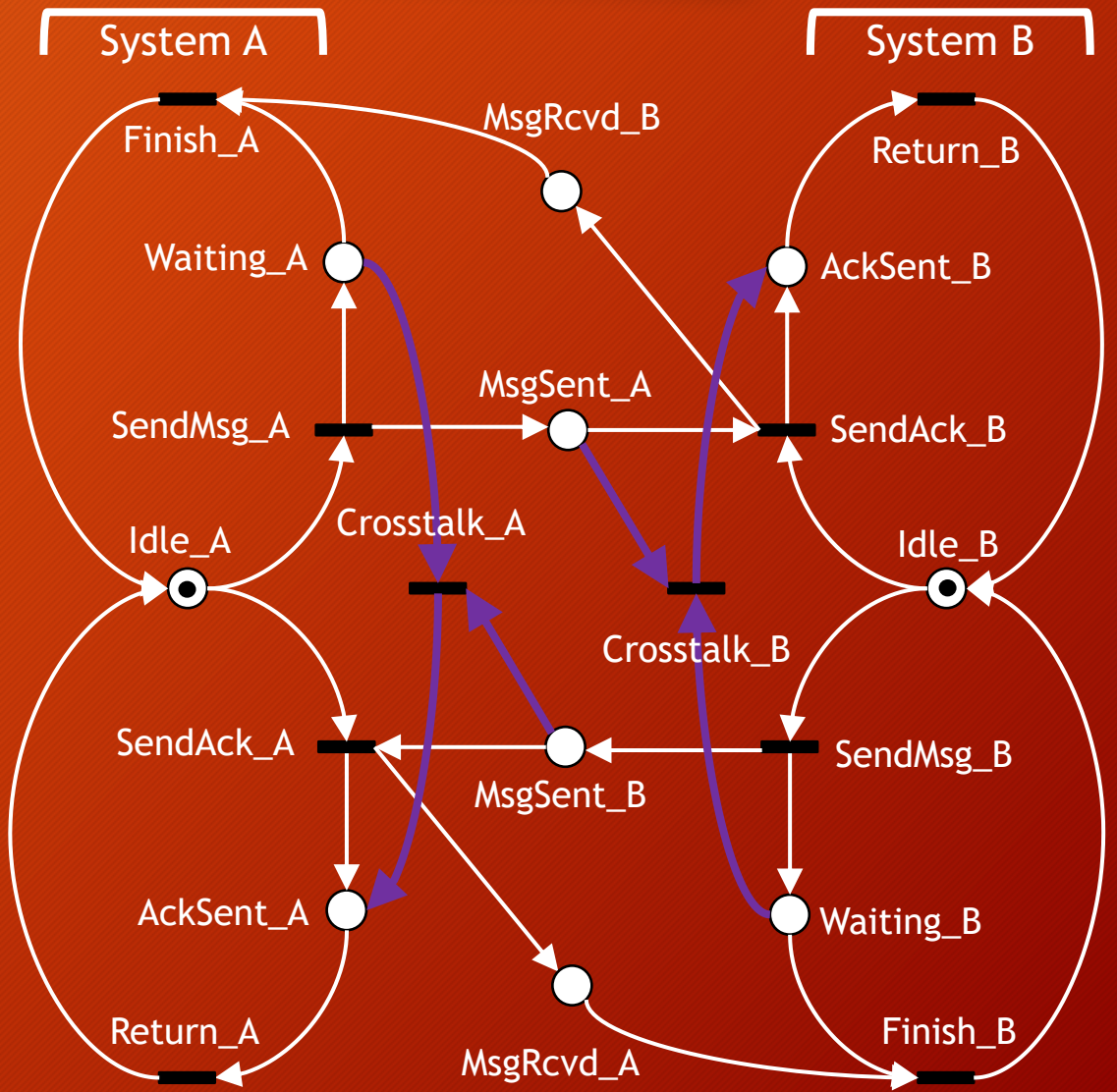
In addition, two communication channels would be required to allow both systems to transmit at the same time

- Crosstalk occurs when the two systems both send messages at the same time, while also waiting for acknowledgement from the other system



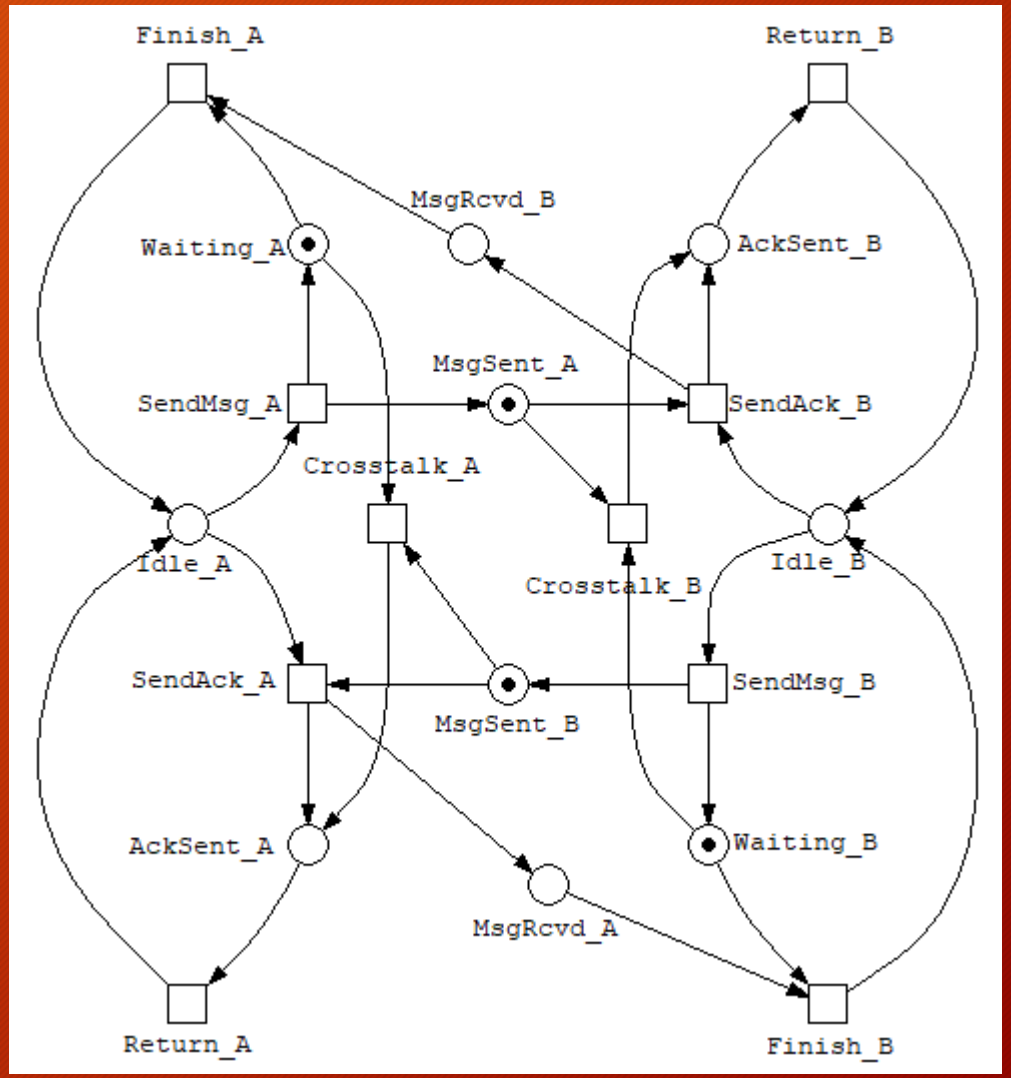
Attempt at Correcting Crosstalk

- One attempt to overcome the crosstalk issue is to catch a situation in which one System is waiting for an ack, while it also is receiving a new message from the other system
- This is crosstalk
- A special transition (Crosstalk) is added to terminate the waiting for the ack from the first received message



Crosstalk Correction in Action

- Here, a simulation run in SNOOPY shows the crosstalk detection scheme working
- Both systems have sent a message
- And both systems are waiting for acknowledgement of their messages having been received
- The crosstalk detection mechanism is ready to fire to terminate waiting for the acknowledgement, and to then go ahead and process the new incoming message and send out an acknowledgement
- This is a case of the crosstalk mechanism working correctly

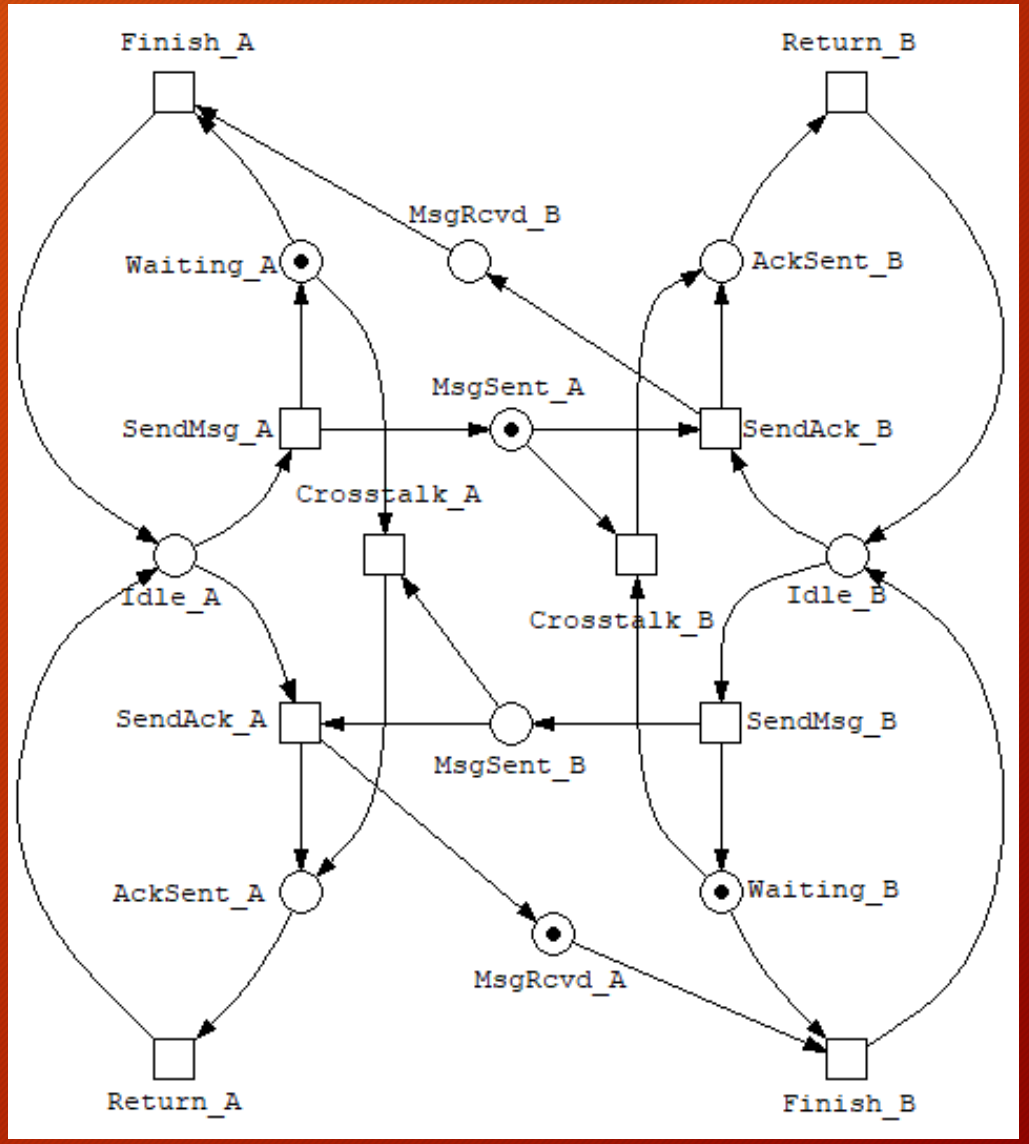


Crosstalk Detection Failure

- However, on occasion, the mechanism declares crosstalk when the situation does not actually exist
- Here, in a simulation run on SNOOPY, System B is waiting on acknowledgement of a message it previously sent to System A
- While System B waits, System A sends a new message
  - This is not a case of crosstalk because both messages are not being sent simultaneously
- Nevertheless, System B is now set to declare a crosstalk (falsely)

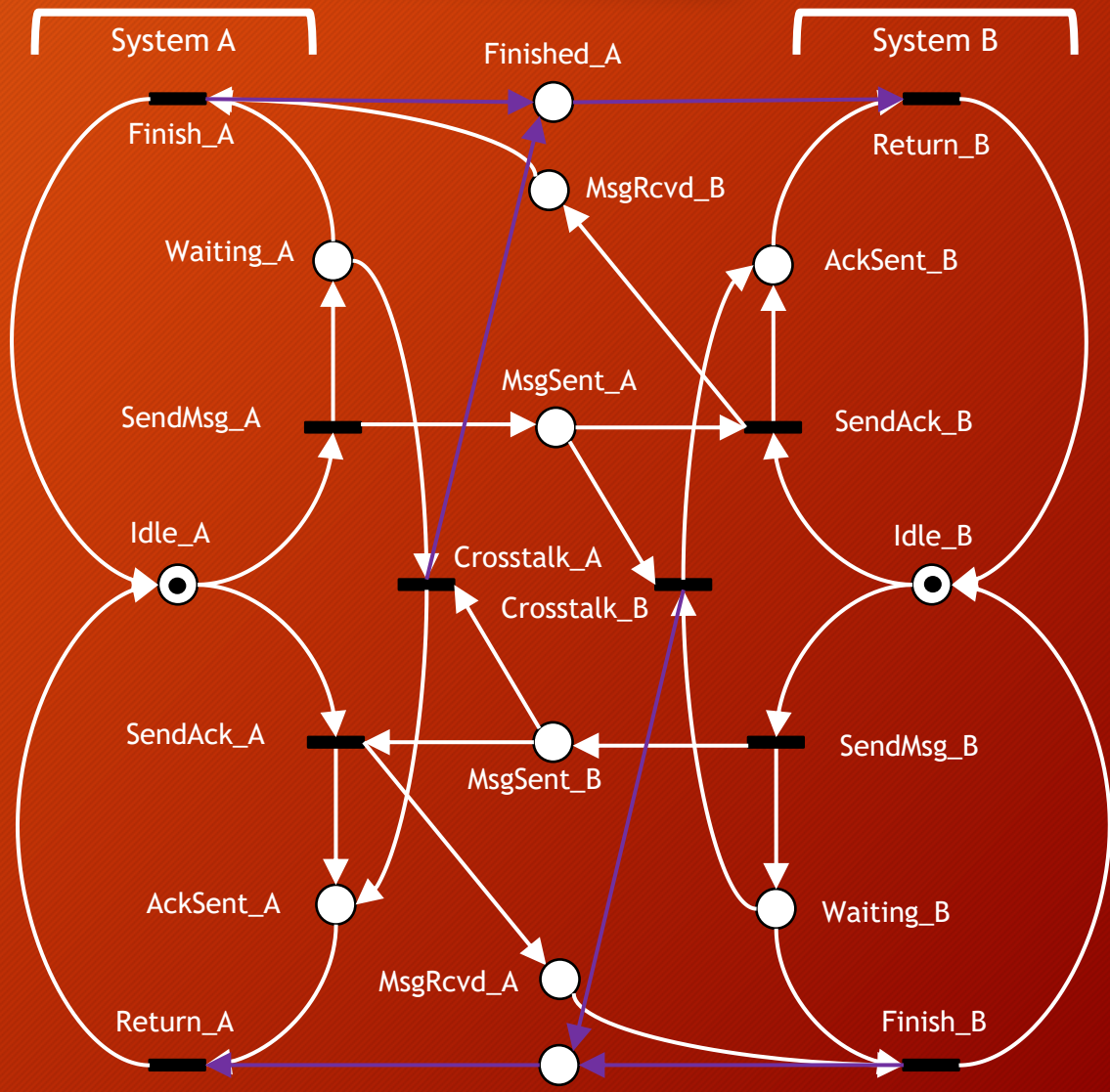
This is a relatively rare occurrence

In this particular case, it took over 100 firings in SNOOPY to encounter this situation



Crosstalk Detection Fixed

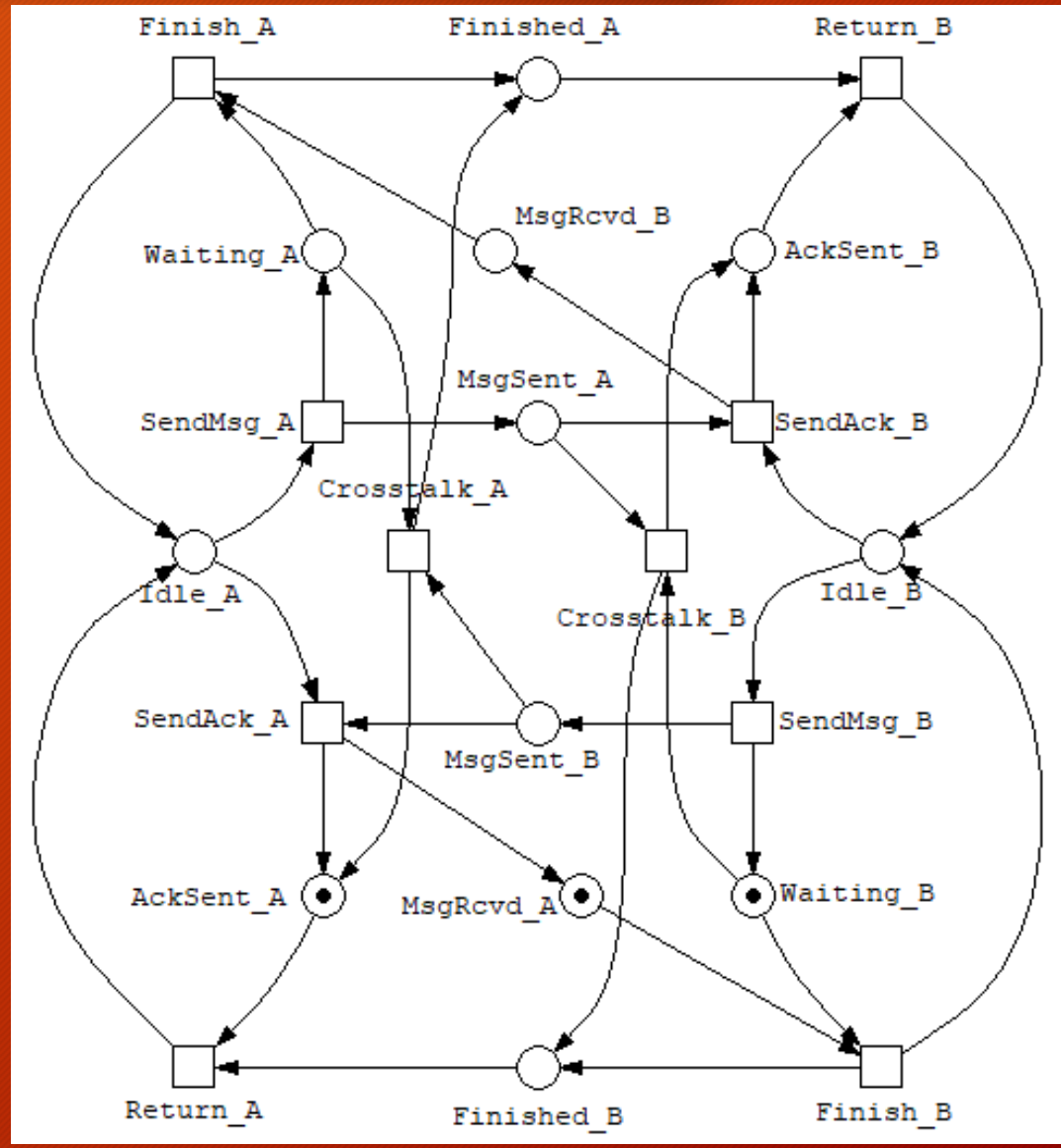
- In this amendment to the network, the problem of false detection of crasstalk is corrected
- Here, a token in AckSent cannot return to Idle until an additional token is provided either by
  - System A finishing it's complete message sending operation after receiving an acknowledgement from System B, or
  - System A recognizing a valid crosstalk situation
- Eitherway, System B will bot be able to initiate a new message (from Idle) until System A completely finishes its processing of the last message sent by System B



Crosstalk Detection Fixed (continued)

- Here is a repeat of the previous scenario in which System B is waiting for an acknowledgement from System B
- System A will NOT be able to return to Idle and send another message UNTIL System A completes its processing of the previous message from System B

This is a complete solution that correctly captures real crosstalk situations and does not capture false croostalk situations



References

1. Reisig, W.(1982). *Petri Nets – An Introduction.* Springer-Verlag, Berlin, Heidelberg, New York
2. Reisig, W. (2013) *Understanding Petri Nets – Modeling Techniques, Analysis Methods, Case Studies.* Springer-Verlag, Berlin, Heidelberg