# Modeling with Petri nets

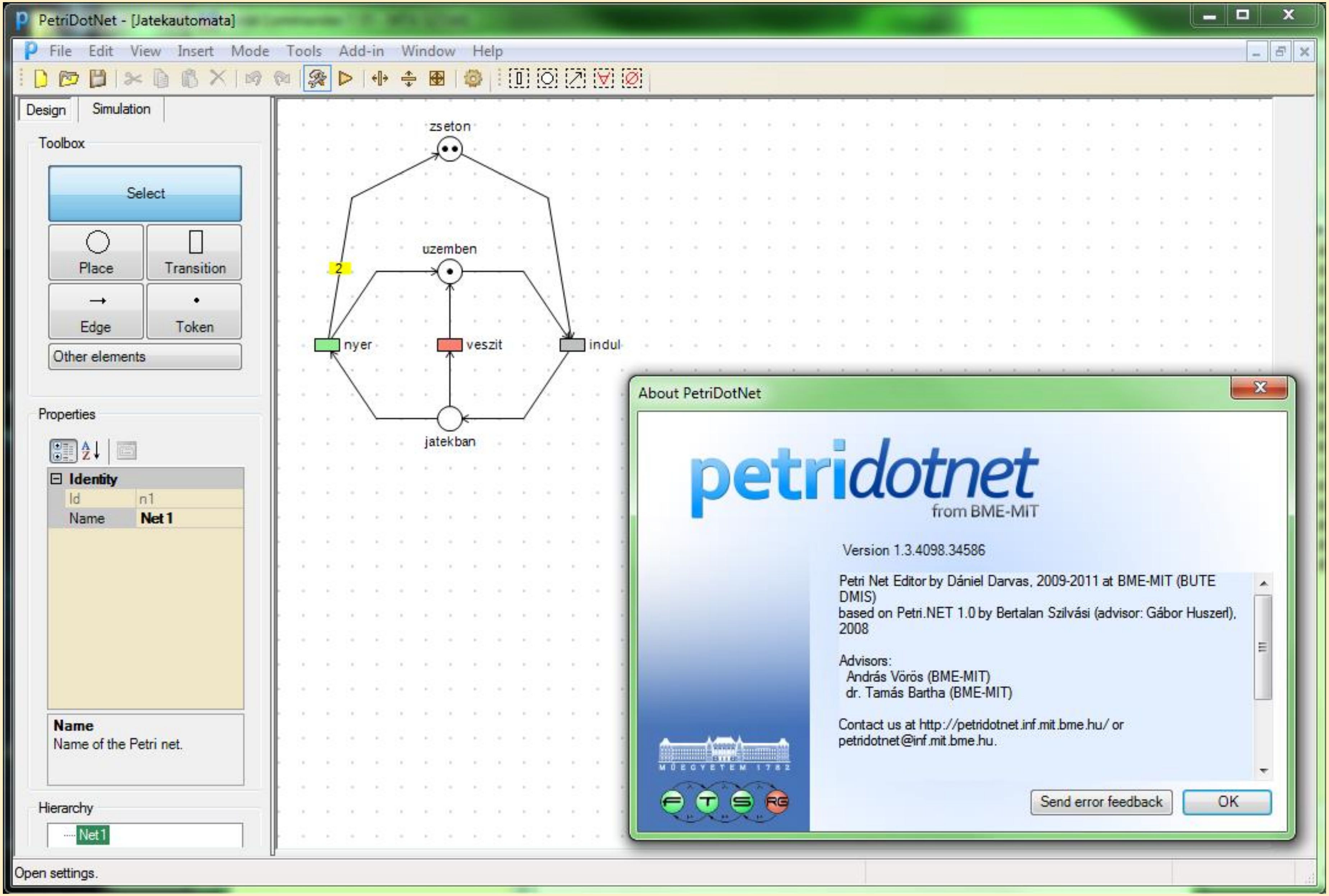
dr. Tamás Bartha
dr. András Pataricza
dr. István Majzik
BME Department of Measurement and Information Systems

# Modeling tools:
# DNAnet, Snoopy, PetriDotNet

# The PetriDotNet modeling tool

- Features
  - Graphical editor + token game + simulation
  - Easy to use, many convenience functions
  - Extensions: inhibitor arcs, timings, colored nets
  - Supports hierarchical Petri nets
  - Supports plug-ins, e.g. analysis modules
  - Dynamic properties, CTL model checker
  - Coloring, rotating elements, displaying arc weights
  - Standard PNML format, with INA export
- Developed by us: petridotnet.inf.mit.bme.hu

# PetriDotNet screenshot



# PetriDotNet analysis features

**Properties of Net AlterBit**

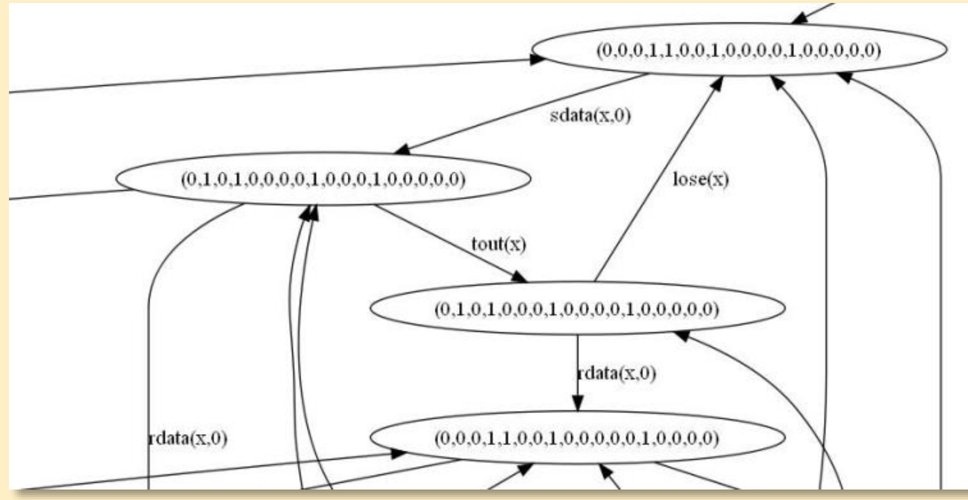
**Dynamic Properties**

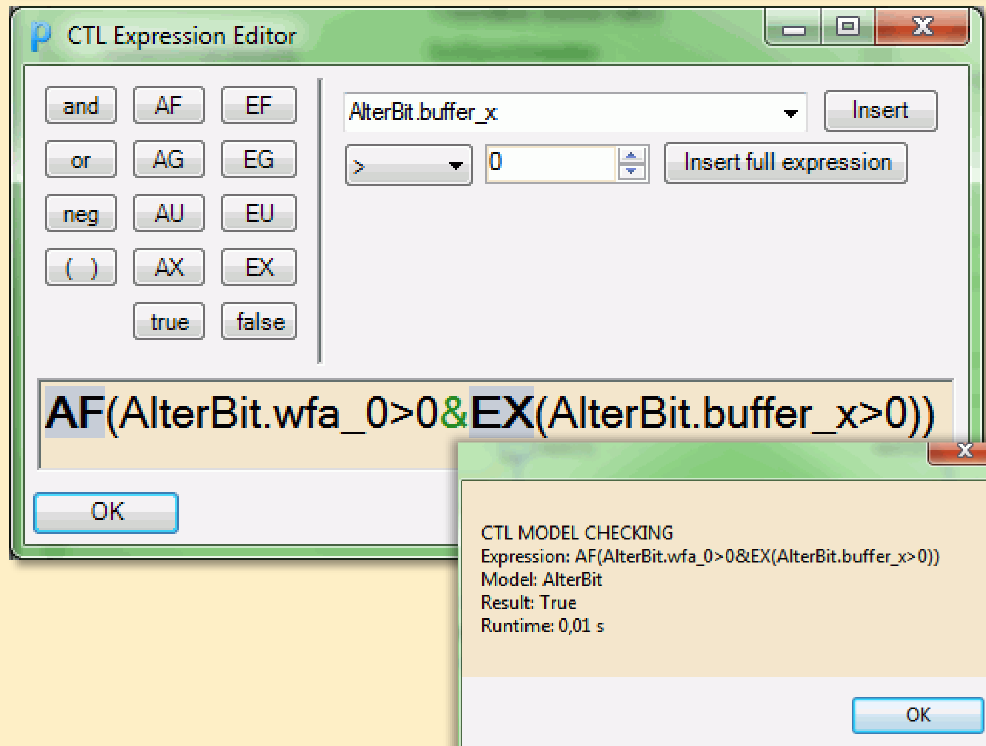
| | |
|---|---|
| Number of states: | 108 |
| Boundedness: | Bounded |
| | 1-bounded (safe net) |
| Deadlock freedom: | Deadlock free |
| Reversibility: | Reversible |
| Persistency: | Non-persistent |

**Static Properties**

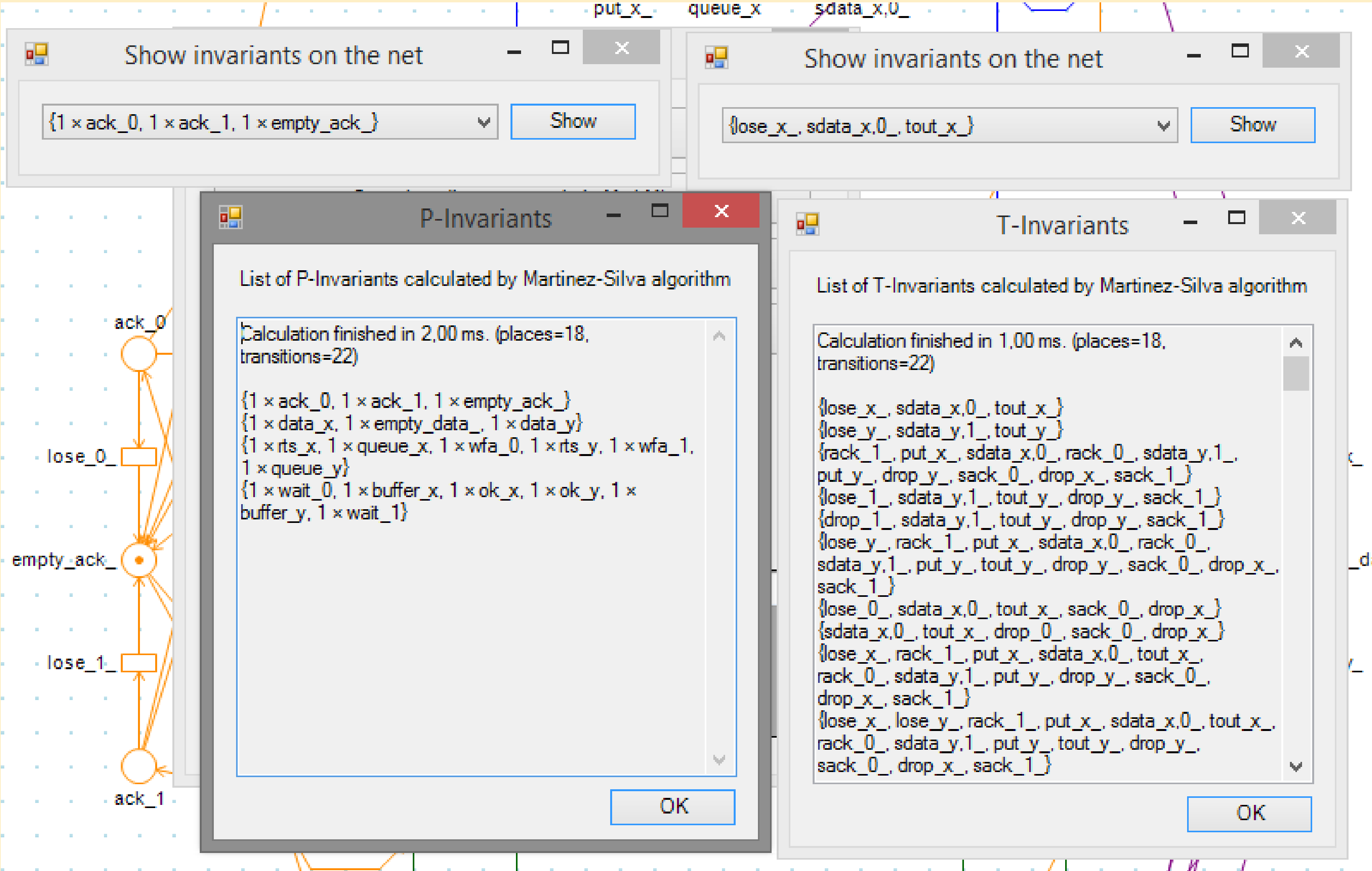
| | |
|---|---|
| Most specific subclass: | Petri Net |
| Purness: | Not Pure |

Reachability check; CTL check; Save the reachability graph; Save adjacency matrix; Search T-invariants; Search P-invariants; Display token bounds of places;



# PetriDotNet invariant analysis



# Basic principles of modeling

# Purpose of system modeling

- IT systems are usually well decomposed
  - Building systems by integrating components
    - Steps, processes, threads, …
  - Relationships between basic components:
    - Explicit logical relationship: order, causality
    - Implicit dependency: e.g. using shared resource
- Model-based analysis: qualitative and/or quantitative
  - Qualitative: proving logical correctness
  - Quantitative: performance analysis, reliability, availability, safety analysis

# Building a model

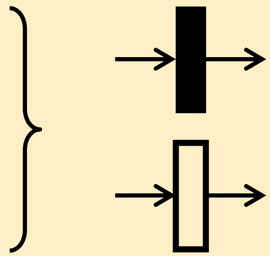
- Three main model element categories:
  - Processes, containing activities
  - Resources (including: data, messages, channels)
  - Interactions between processes and resources
- Modeling: hierarchical and functional
  - Bottom up:
    Basic activities -> (Composite activities ->)
    Subprocesses -> Composite processes
- Steps:
  - Building individual model elements
  - Integration

# Typical steps of system modeling

1. The process model (without detailed resource usage and communication)
2. The resource model
   - A finite automaton part with busy/idle/… states
   - Message queue (if needed)
3. Integration: Fusion of corresponding transitions in the process and resource models
   - E.g.: *Occupying* fused with transition $Idle \rightarrow Busy$
   - E.g.: sending message puts message into queue

# Modeling activities in Petri nets

- Basic activity: firing a transition
- Resources used: input / output places
- Execution time
  - deterministic
  - stochastic



deterministically timed transition

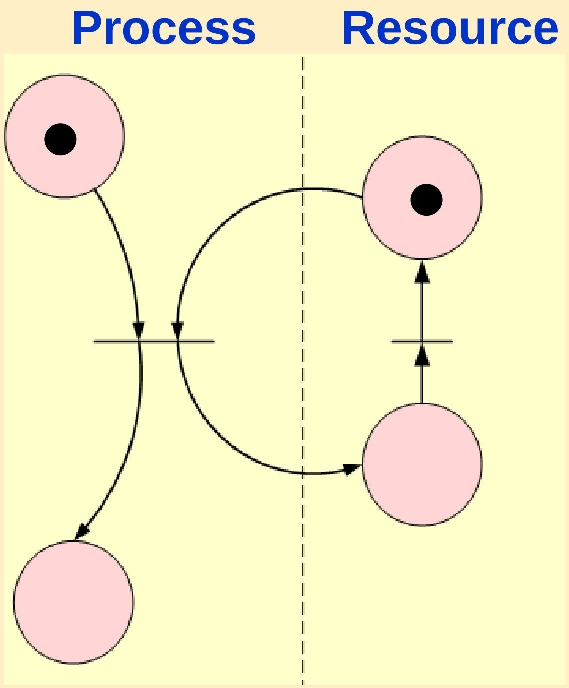
exponentially timed transition

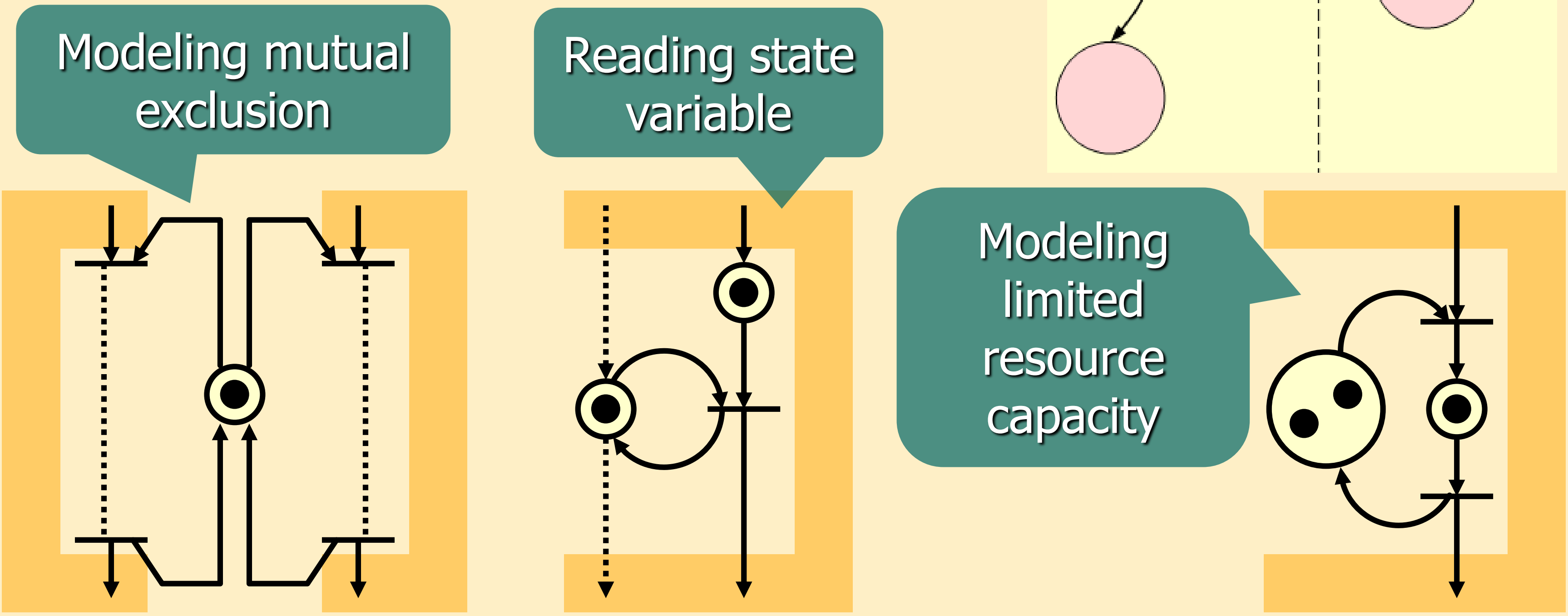
Questions regarding enabledness:

- Untimed transitions fire first (higher “priority”)
- What happens with time after becoming disabled?
  - Restarts (new random): “restarts” activity
  - Remains (previous time): “continues” activity

# Example: Modeling resource allocation

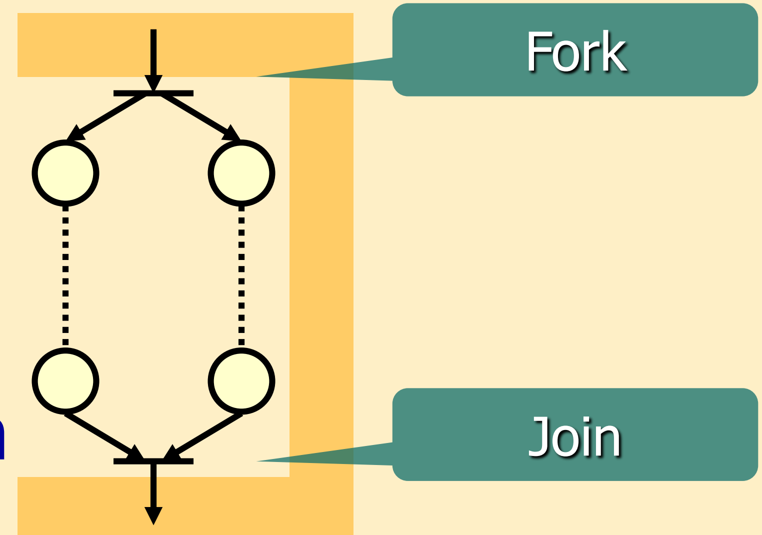
- Allocation of required resource
- Mutual exclusion
- Using limited resource

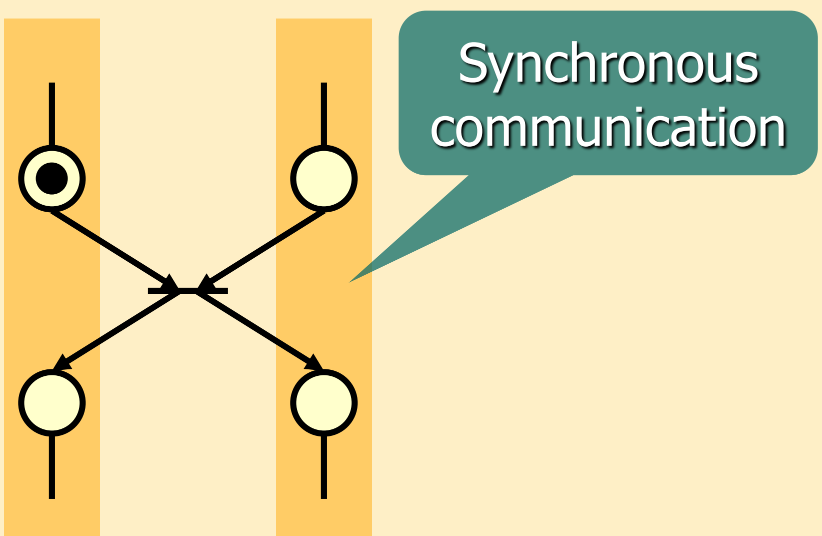


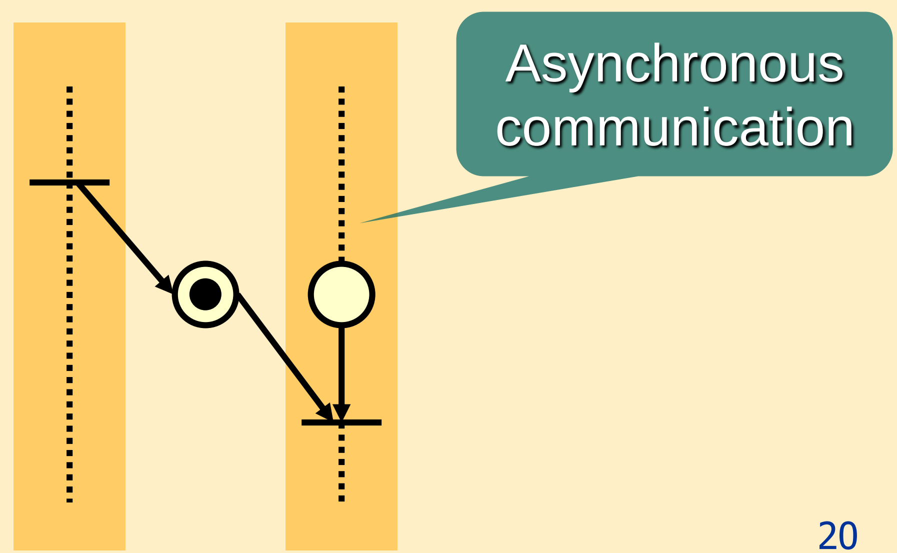



# Example: Relationships between processes

- Parallelism
  - Fork and join
- Synchronous communication
  - Wait for the other
- Asynchronous communication
  - Like a mailbox



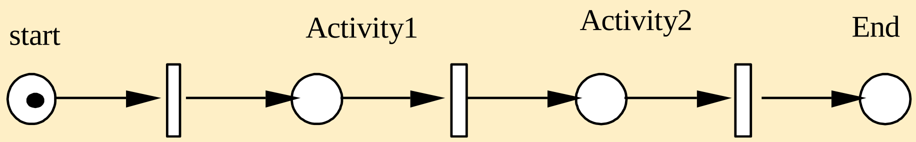



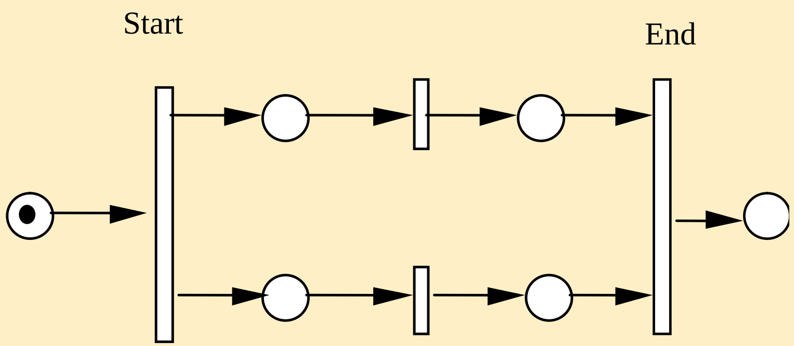
# Example: Modeling a production cell

# Processors

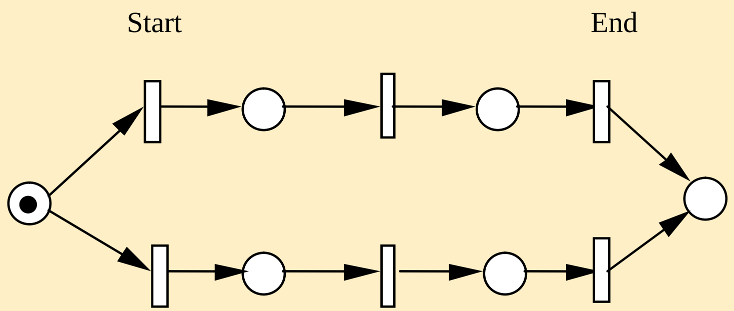
- Sequential processors:




- Parallel processor:

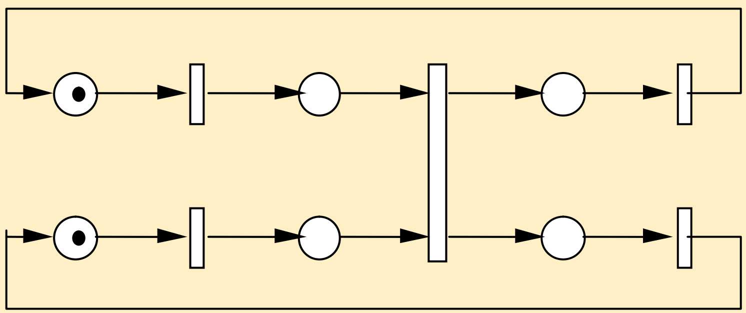



- Alternative processor:

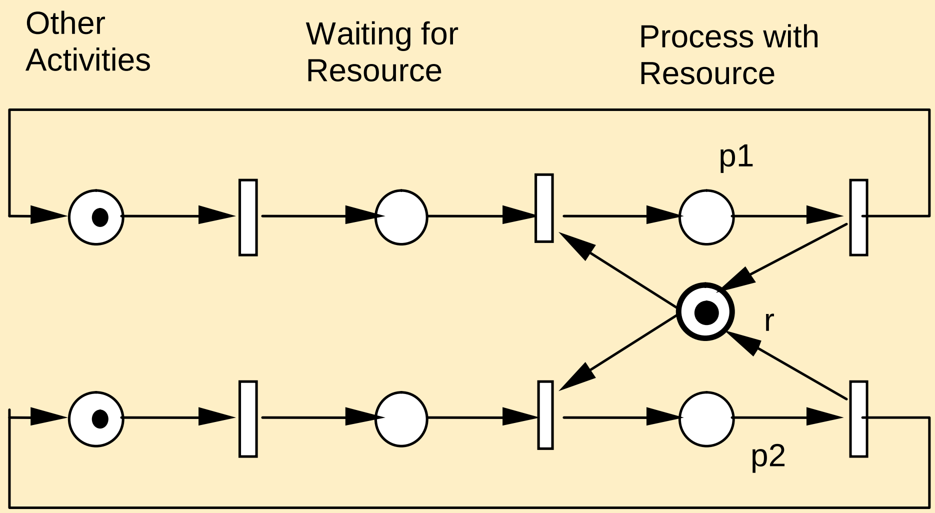


# Interactions

- Synchronization:

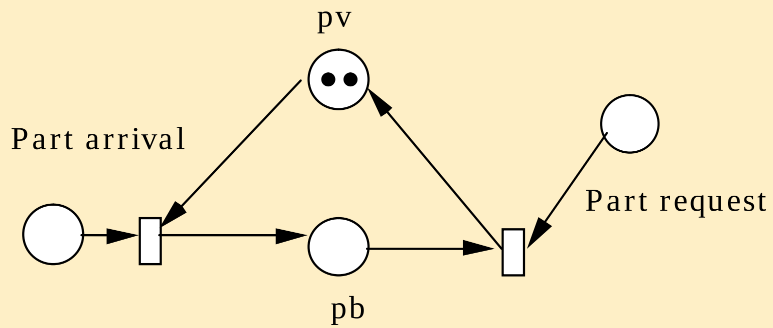


- Shared resource:

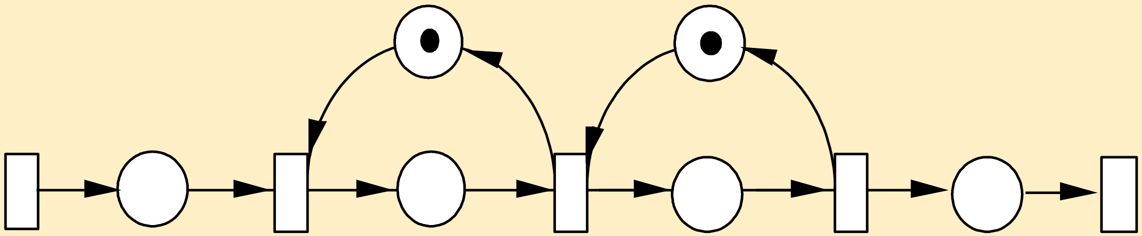


# Containers for processors

- Bounded capacity container:

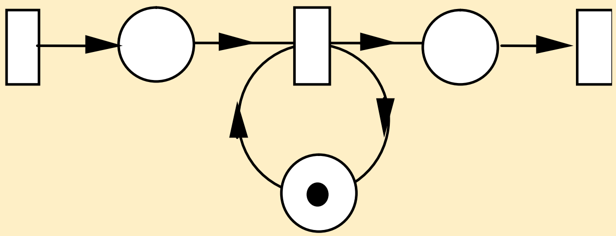



- FIFO container:

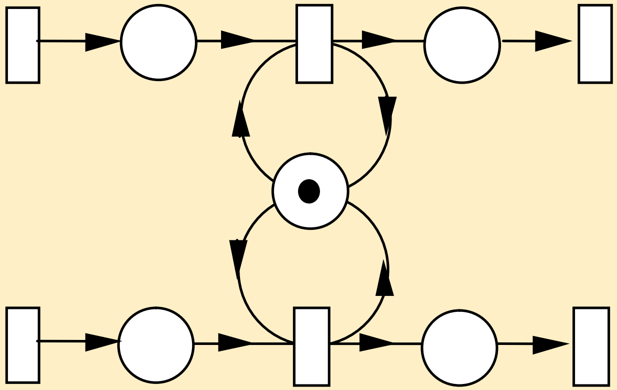


# Using machines

- Process with dedicated machine:

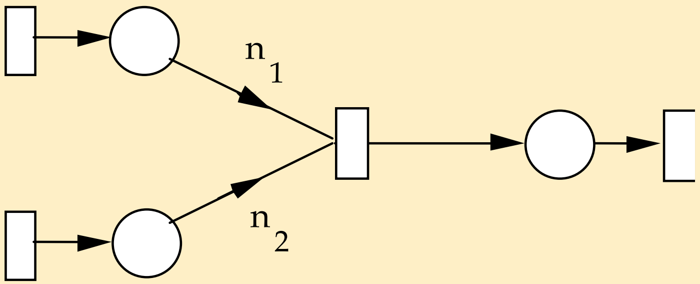


- Process with shared machine:

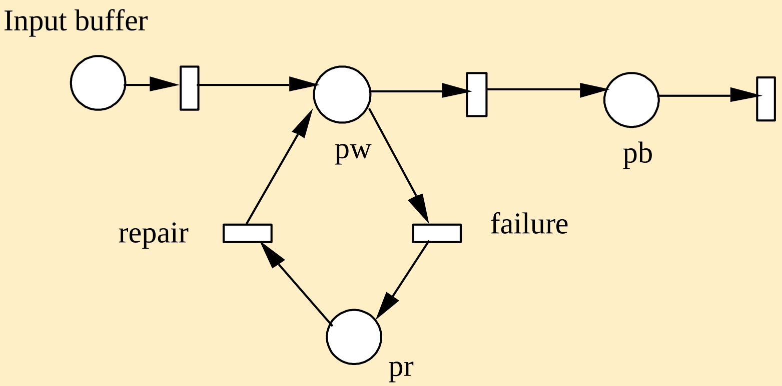


# Assembly

- Assembling parts:

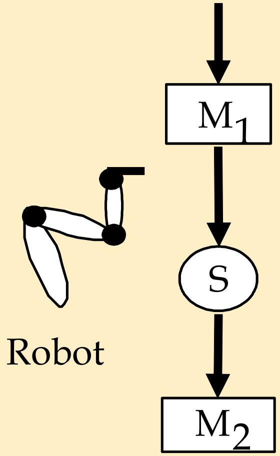


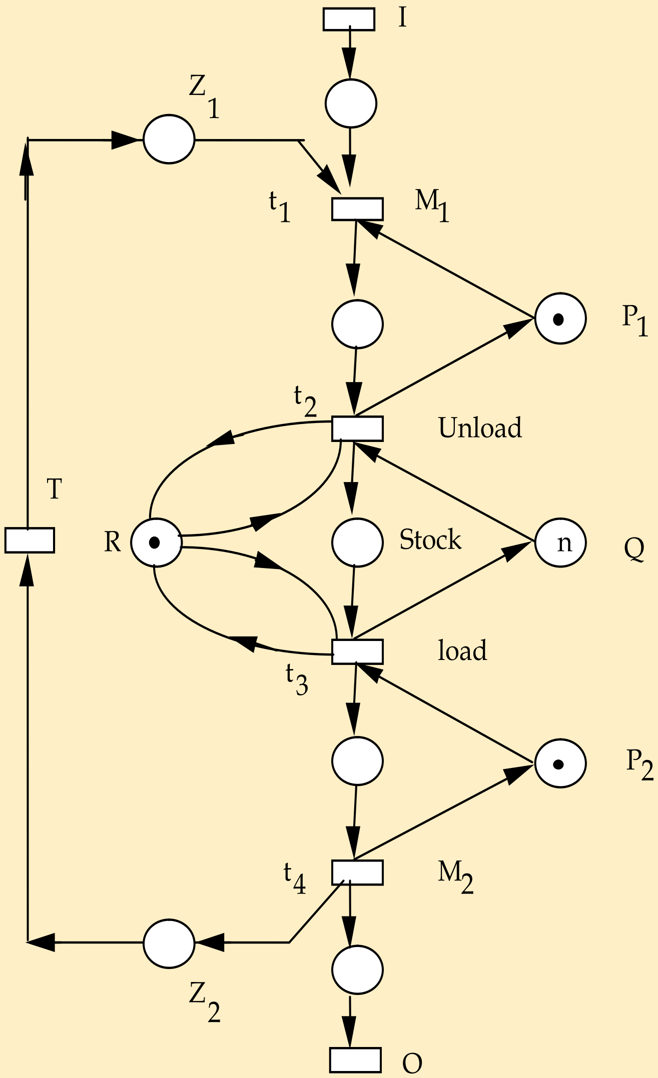

- Failure during process:



# Robot cell

- Activities
- Containers (bounded capacity)
- Resources
- Cyclic behavior



# Example Petri net:
# Alternating bit protocol

# The modeling task

## Alternating Bit Protocol

- Transmission protocol for faulty channels
  - Messages can get lost (a finite number of times)
  - Contents of messages cannot change
- Goal: the protocol should ensure (with a bounded number of steps) that the message is transmitted to the receiver

# Sender process

- Attaches a checking bit to the message
- Received messages are confirmed by the receiver, with the same checking bit
- If the bit attached to the message is $\mathbf{b^0}$, then
  - if the message is lost, the sender detects the lack of confirmation with a timeout $\rightarrow$ sends again
  - if the sender receives a confirmation with a bit $\mathbf{b^0}$ (which is expected), then a negated bit is attached $\mathbf{b^1} = \neg\, \mathbf{b^0}$ to the next message
  - if the sender receives a confirmation with a bit $\mathbf{b^1} = \neg\, \mathbf{b^0}$ (despite expecting $\mathbf{b^0}$), then the confirmation is discarded (and a timeout will occur due to the lack of confirmation)

# Receiving process

- Confirms receiving the message by sending back the same checking bit
- If a message with checking bit $b^0$ is received, then it is confirmed by sending $b^0$ back, then
  - If the bit of the next message is $b^1$ (correct), then sends $b^1$ back to acknowledge
  - If the bit of the next message is $b^0$ (incorrect), then the message is discarded, but sends a confirmation (assuming that it was a repeated message due to the lack of confirmation)

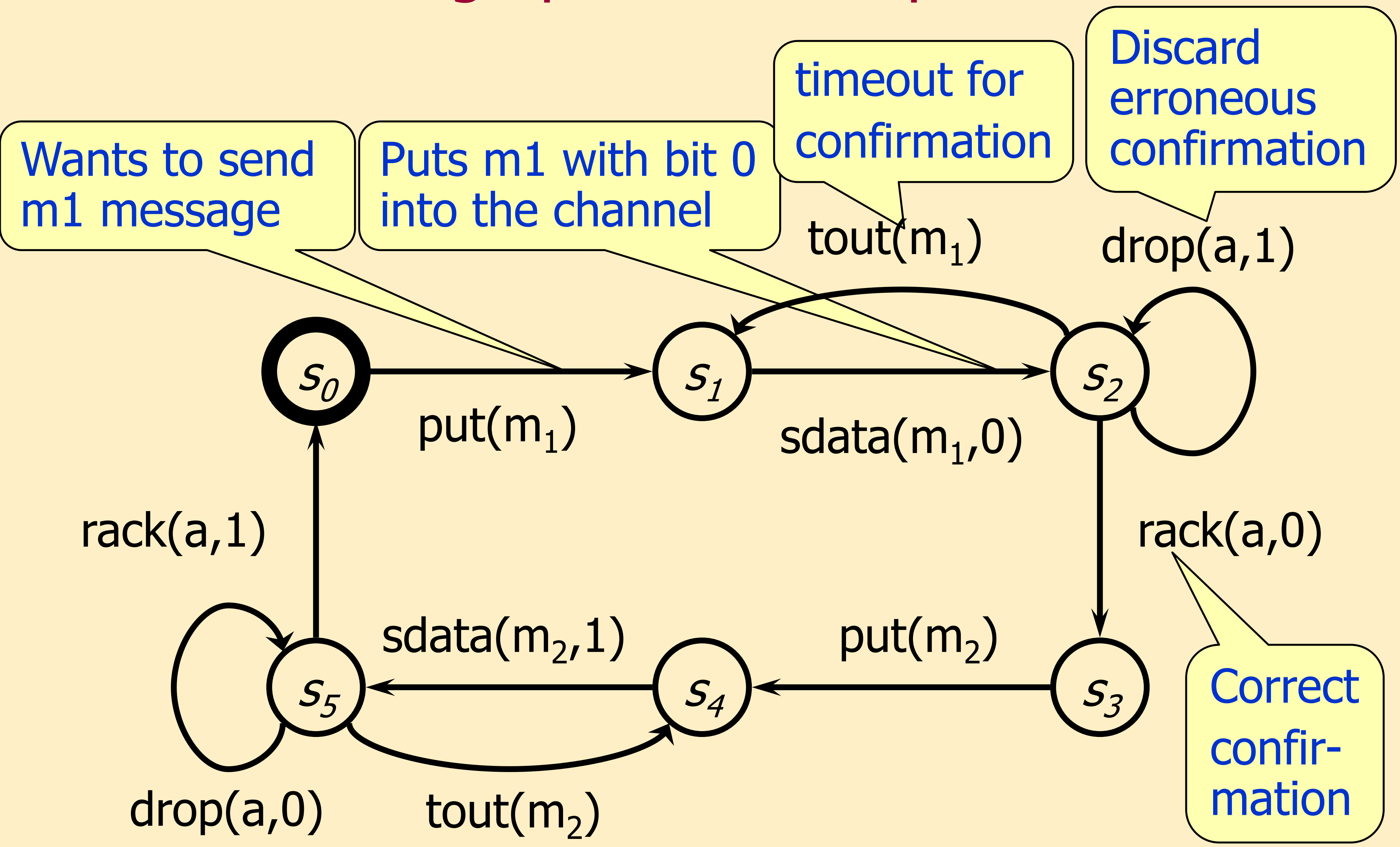
# Steps of building the model

1. Decompose the task to actors and resources
2. Determine the states of actors
3. Determine states of resources and message buffers
4. Create Petri net models from state-based models
5. Integrate actor and resource models
6. Check integrated model
7. Use the model to solve the task

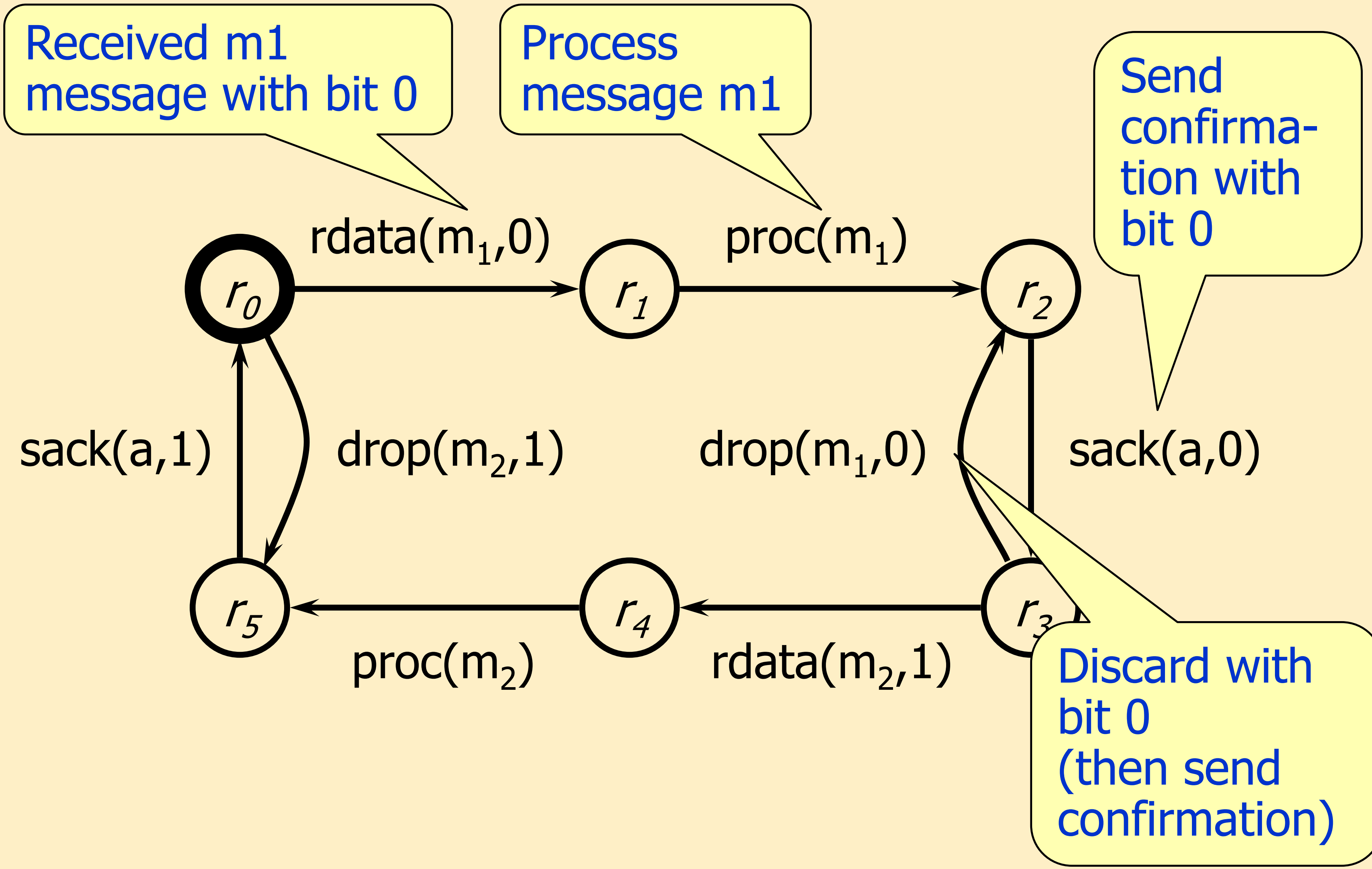
# Components and states

- Components (subsystems)
  - Actors: sender process, receiver process
  - Resources: data channel, confirmation channel
- Each components have its own state
  - State graph: states are circles, events are arcs
- Same events happen at the same time: synchronization

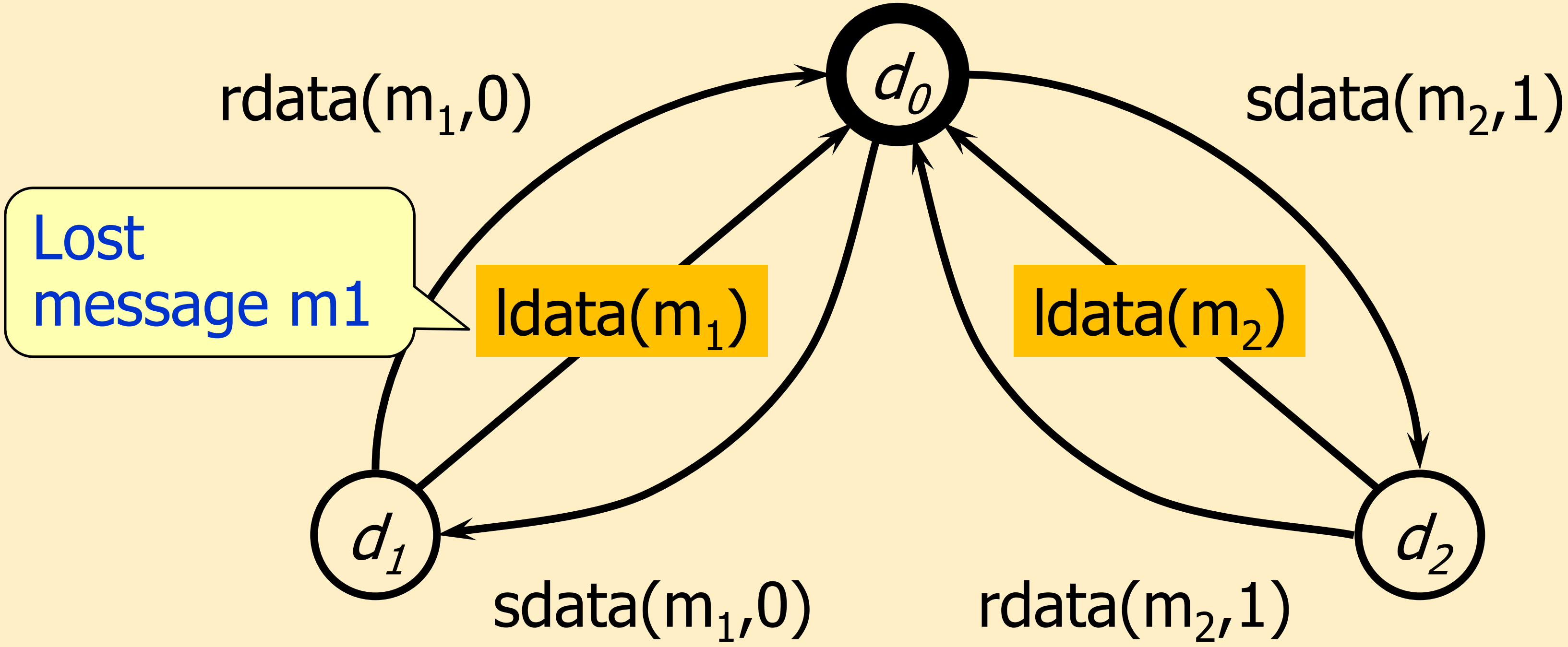
# State graph of sender process



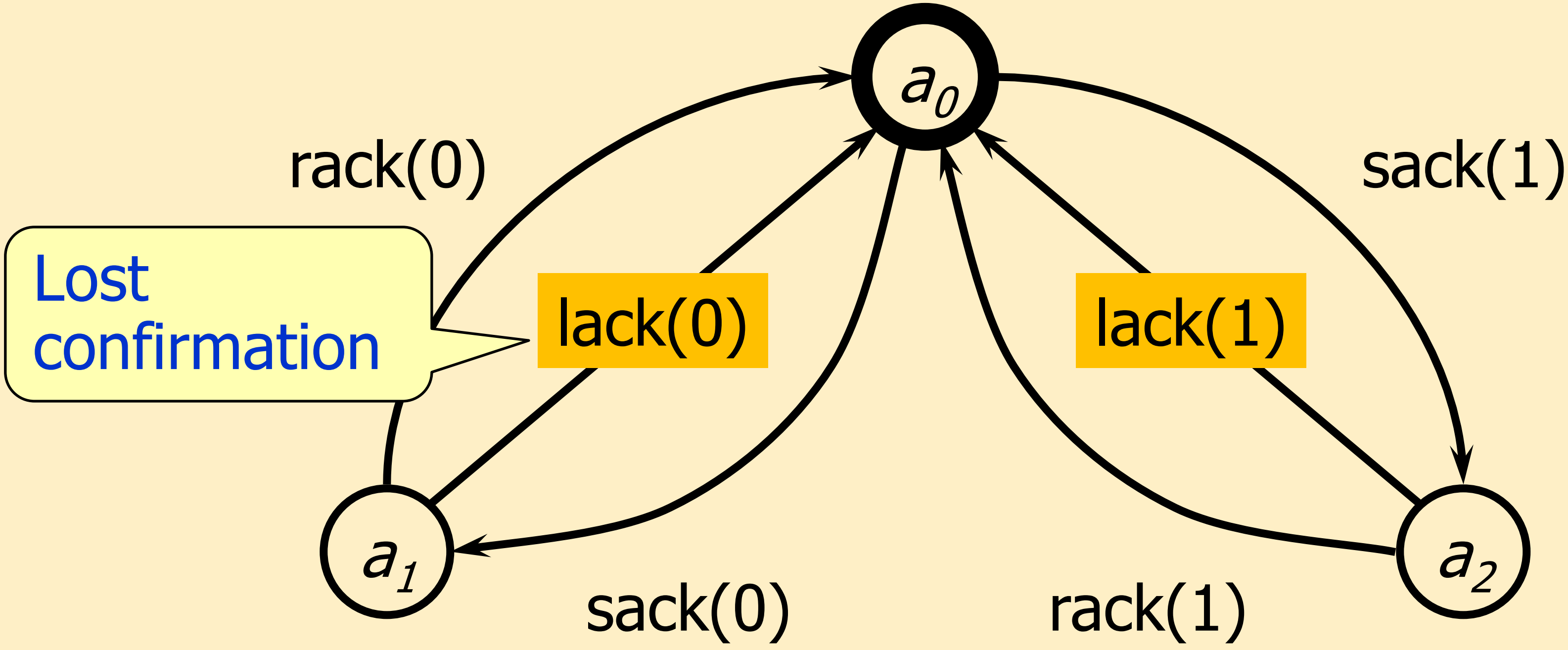
# State graph of receiver process



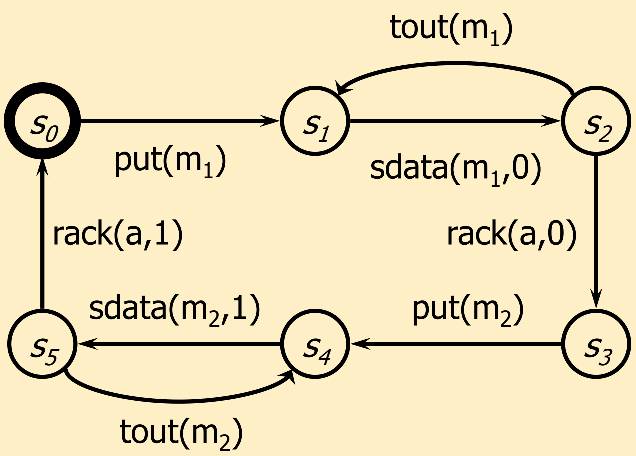
# State graph of data channel

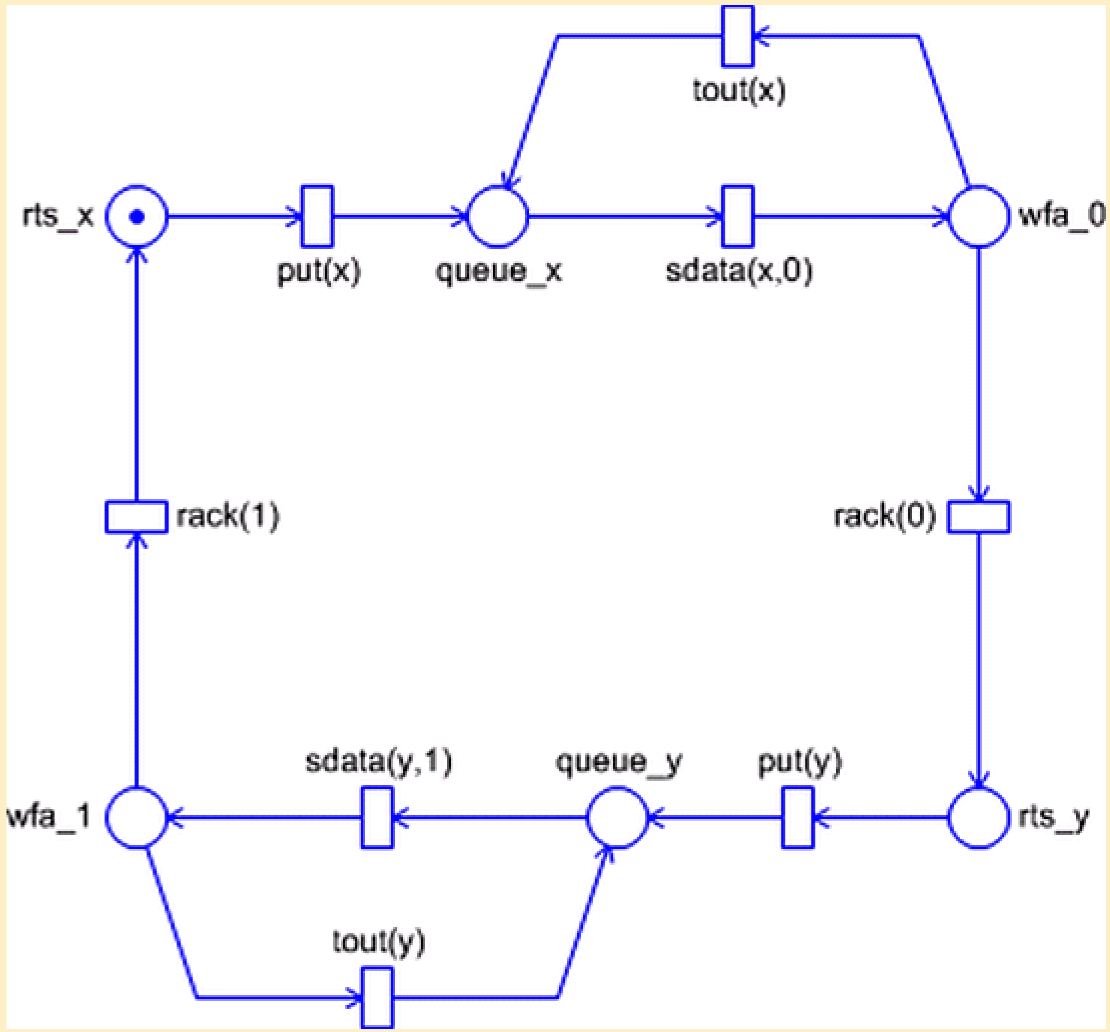


# State graph of confirmation channel

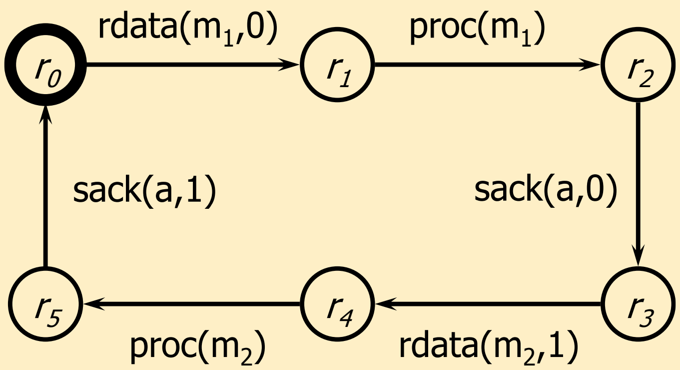


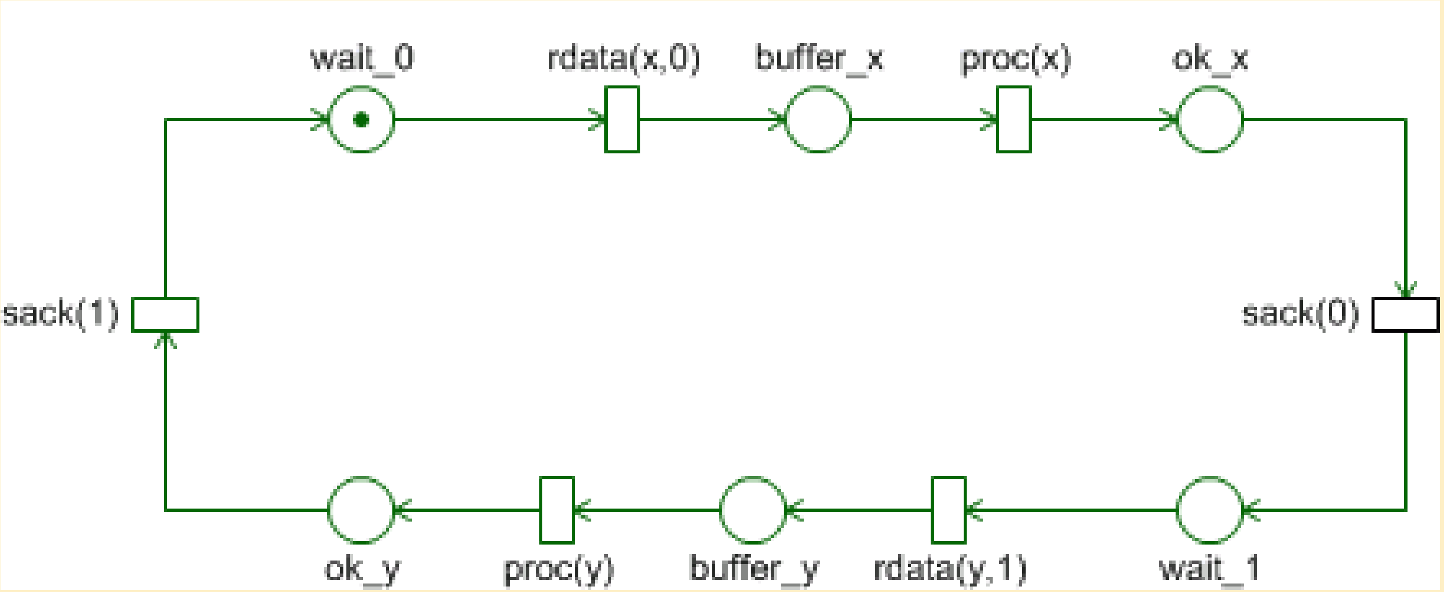
# Petri net model of sender process (main loop)



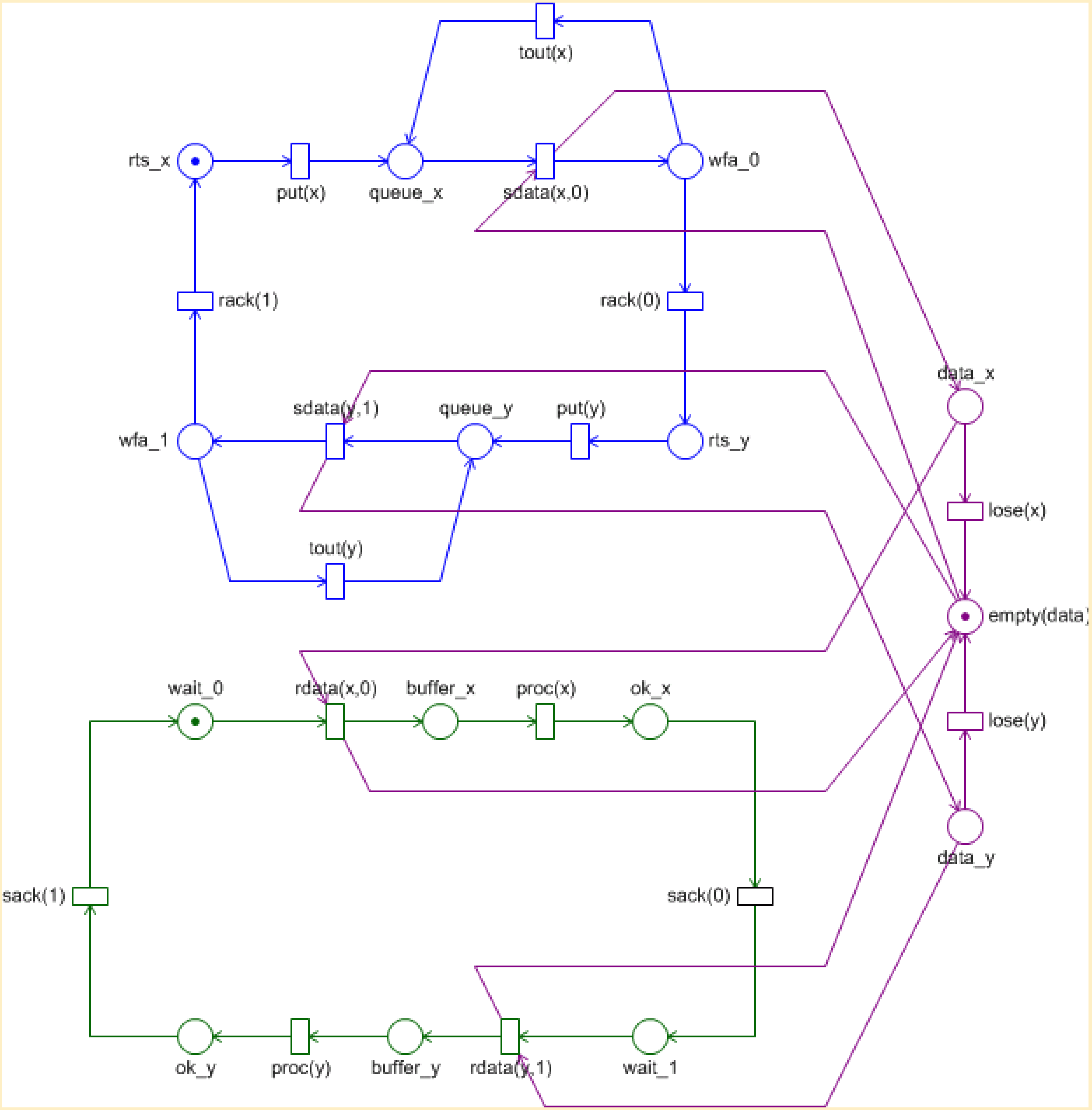



# Petri net model of receiver process (main loop)

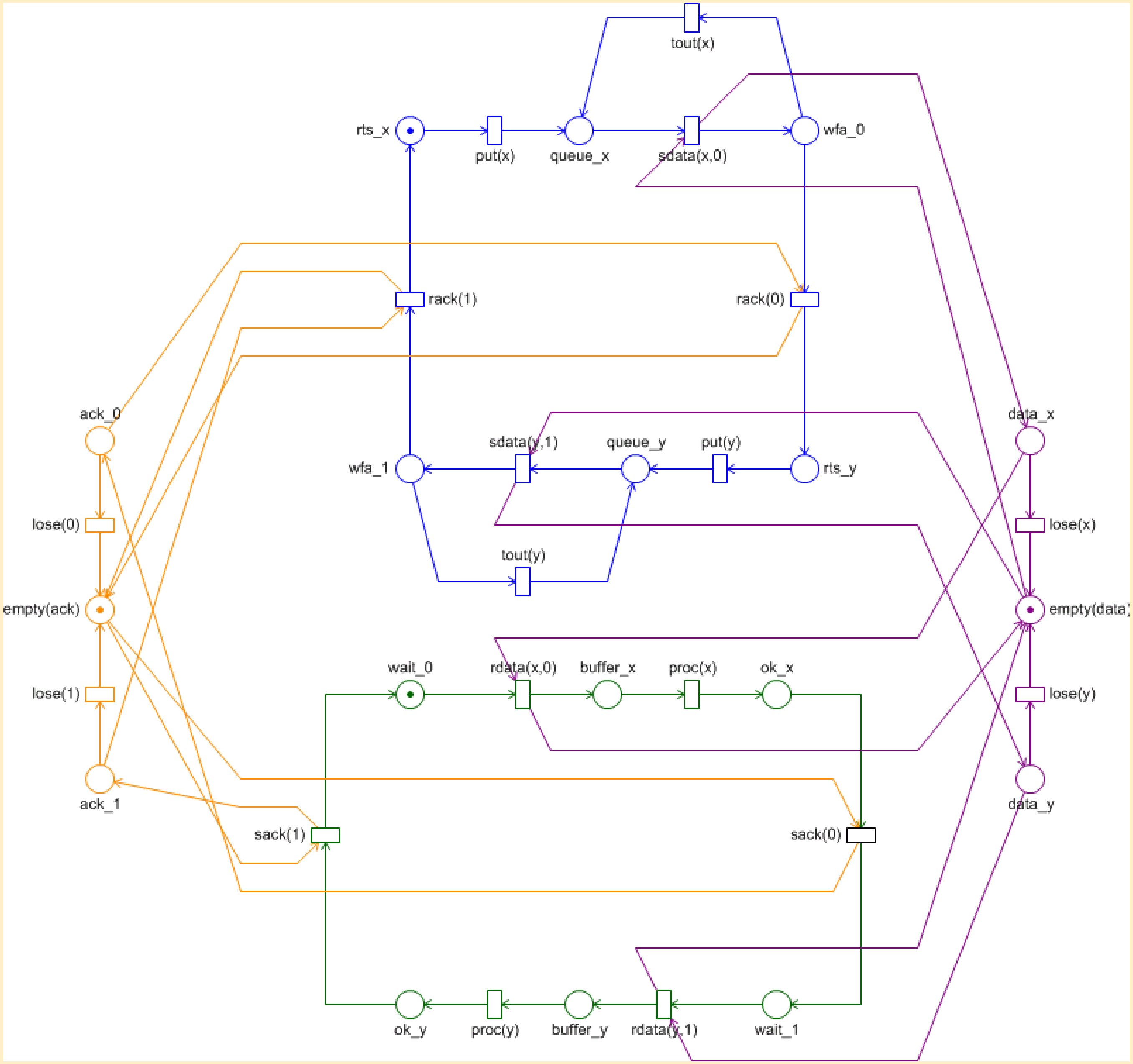


# Data channel and data transmission (main loop)



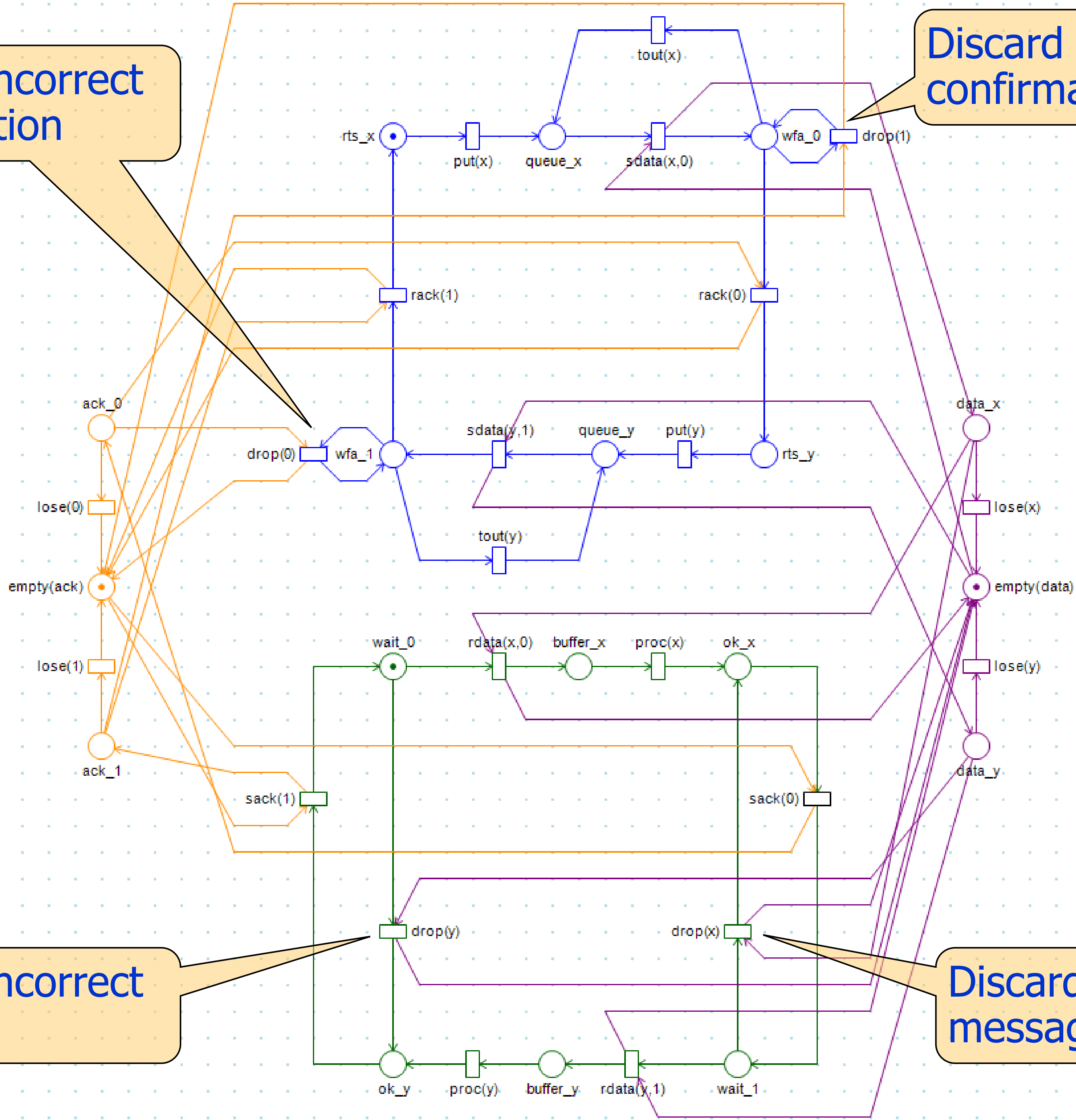
# Confirmation channel and confirmation (main loop)



# Extensions

Discard incorrect confirmation

Discard incorrect confirmation

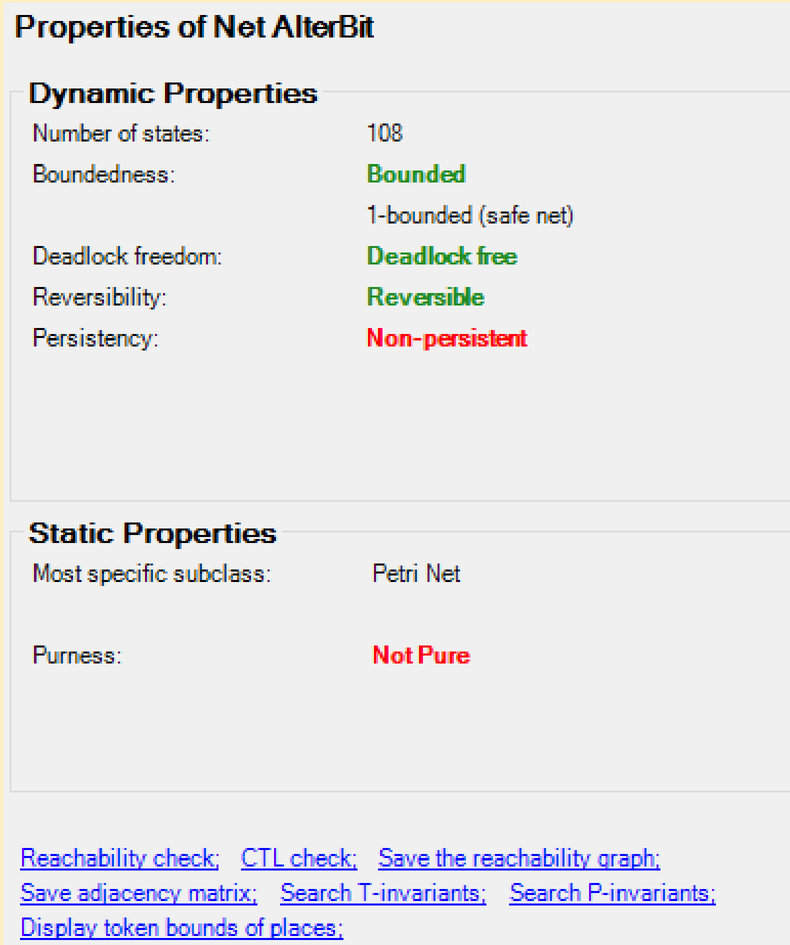



Discard incorrect message

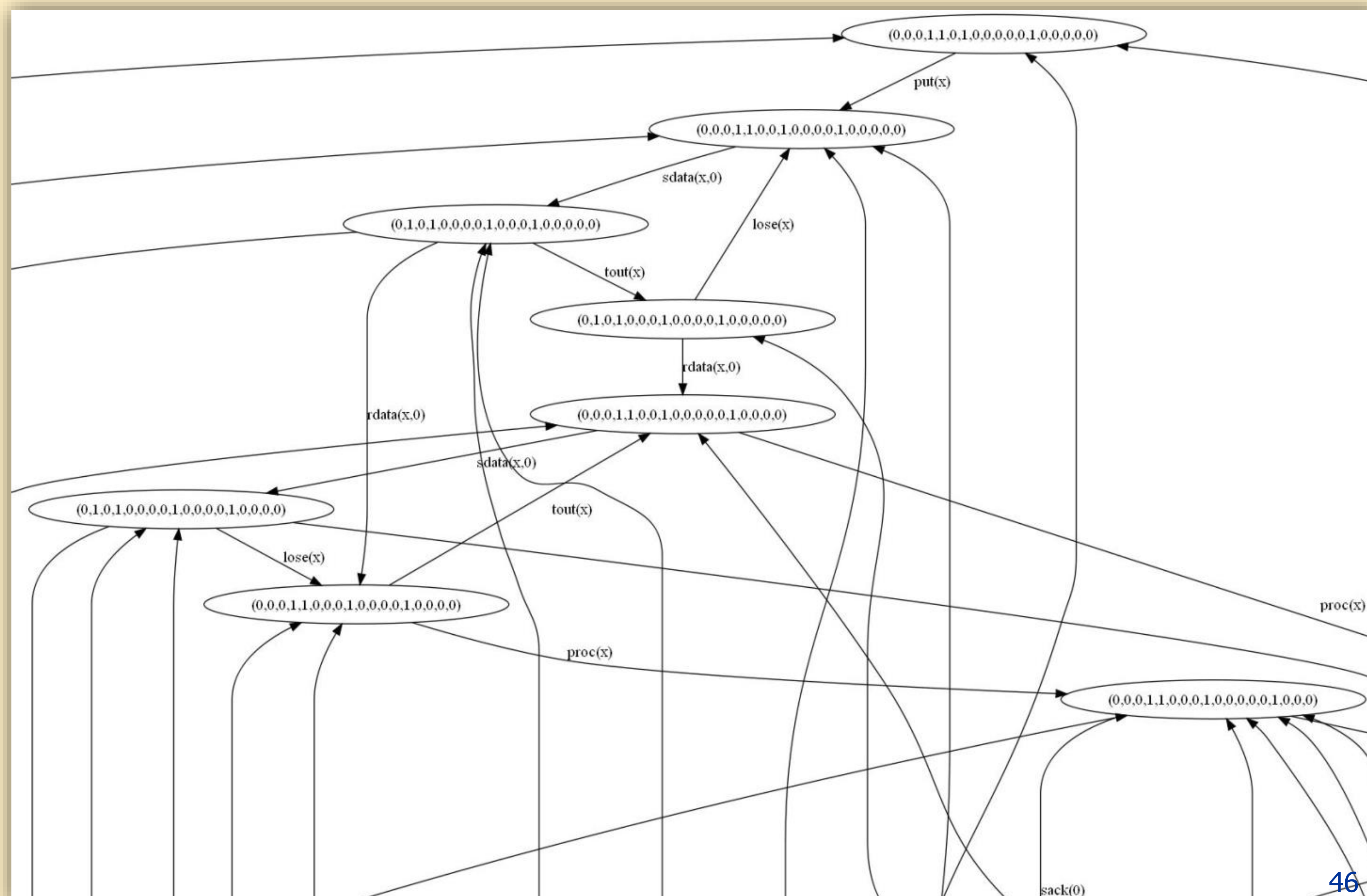
Discard incorrect message

# Example Petri net:
# Alternating bit protocol

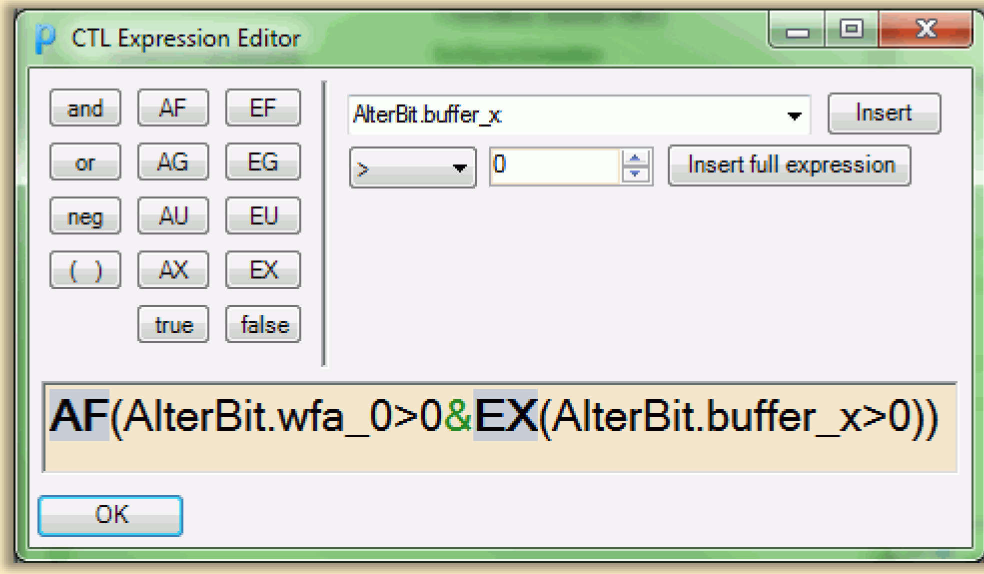
# PetriDotNet: Dynamic properties of the model

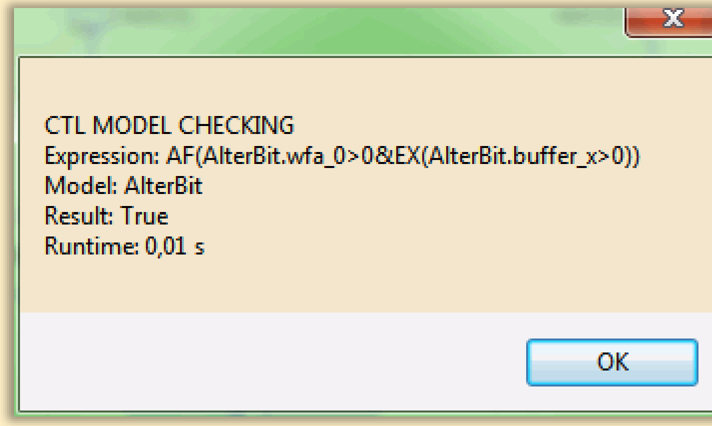


# PetriDotNet: Reachability graph (GraphViz)



# PetriDotNet: CTL model checking




**AF**(AlterBit.wfa_0>0 & **EX**(AlterBit.buffer_x>0)) ⇒ True

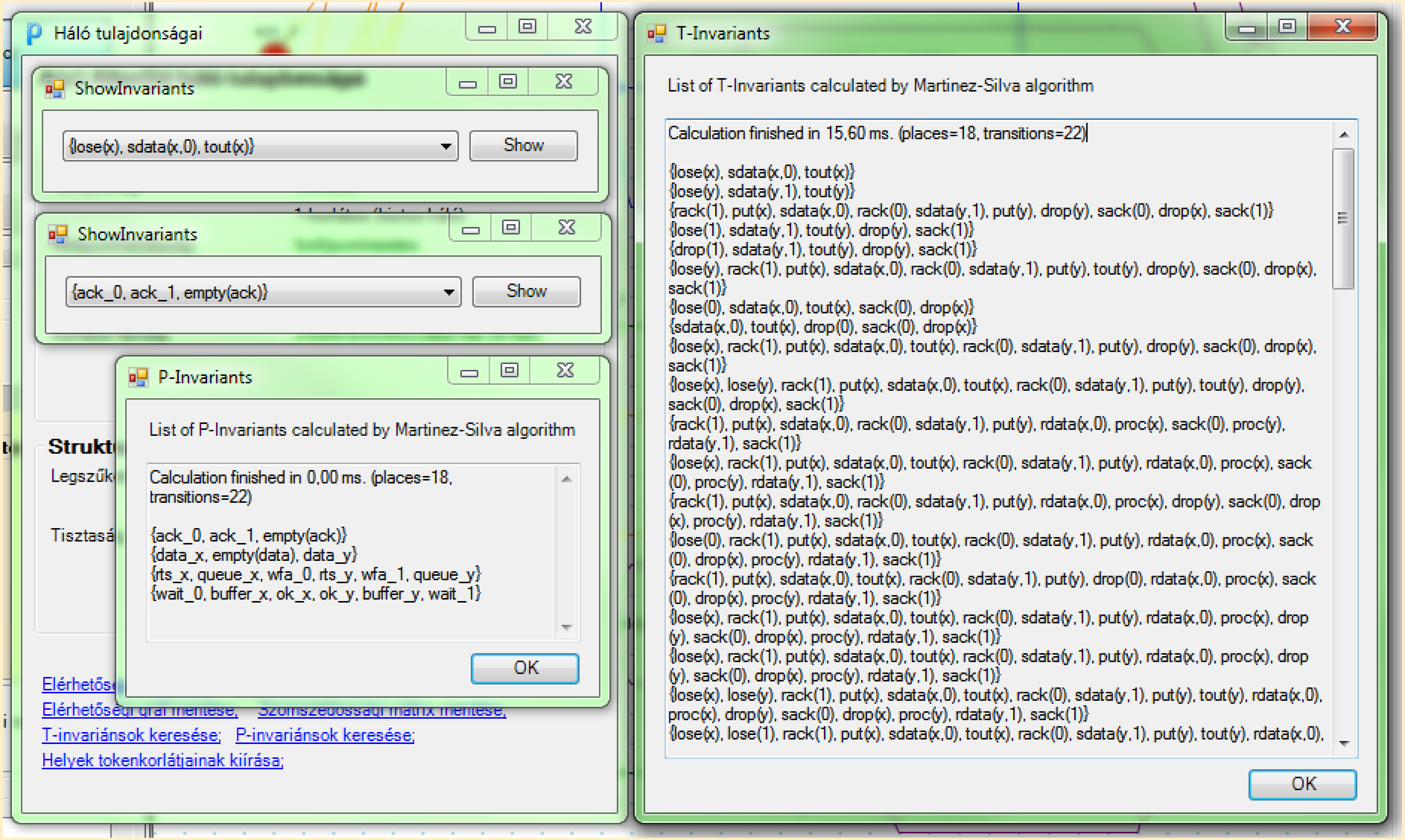
**AG**(**AF**(AlterBit.buffer_y>0)) ⇒ False

**AF**(**EG**(AlterBit.buffer_x=0)) ⇒ True

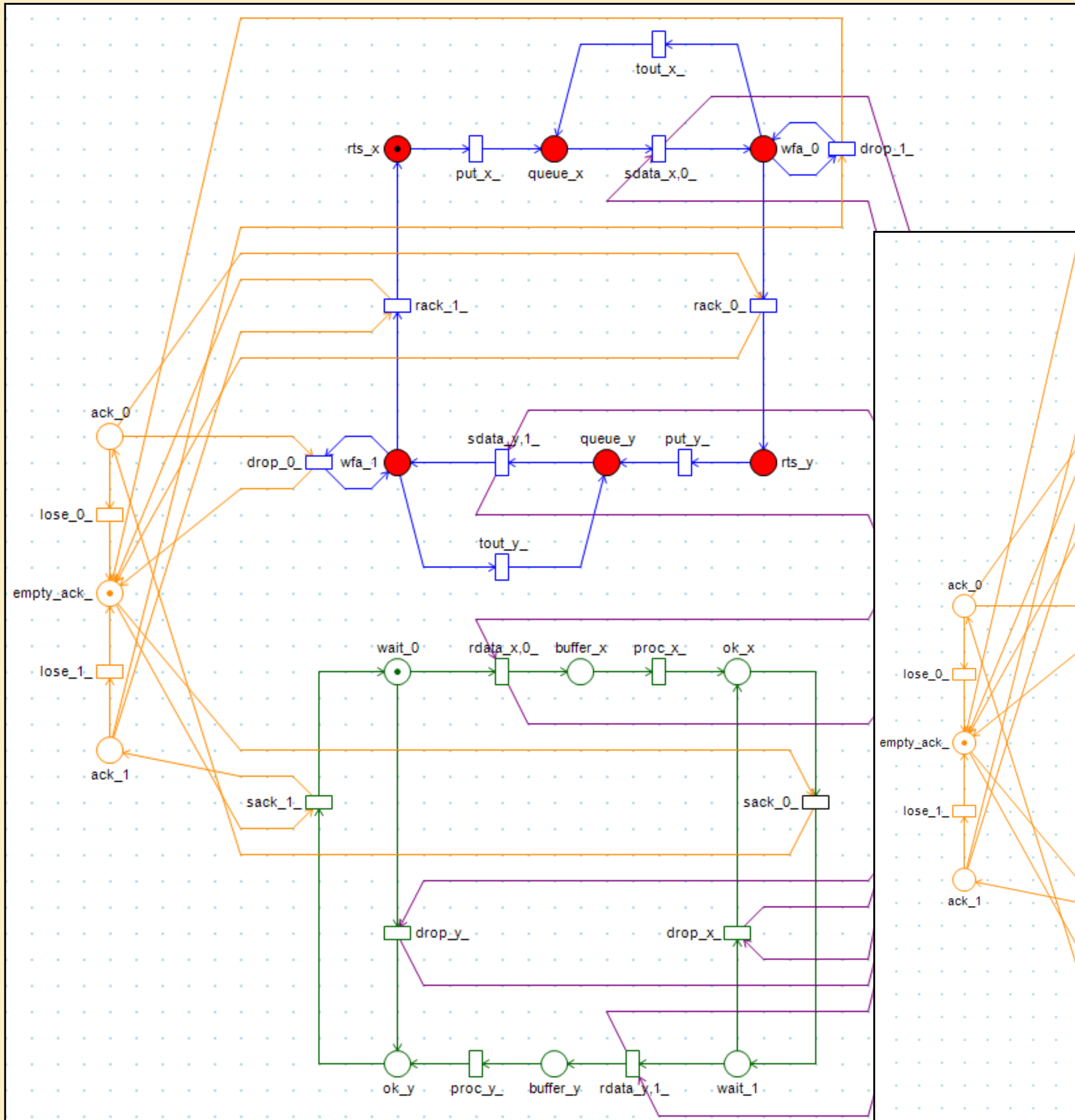
**EF**(AlterBit.wfa_0>0 & AlterBit.data_x=0) ⇒ True

**AF**(AlterBit.queue_x>0 & **AX**(AlterBit.wfa_0>0 & AlterBit.data_x>0)) ⇒ True

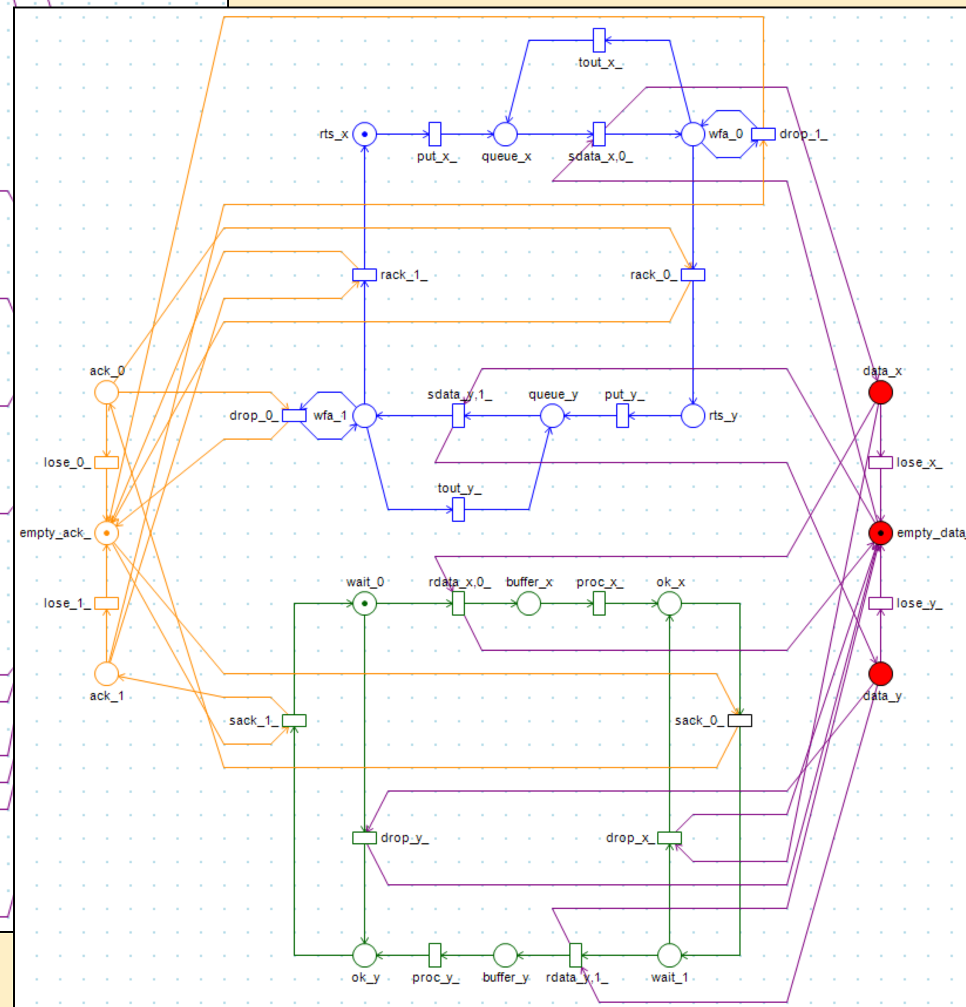
# PetriDotNet: Invariant analysis



# PetriDotNet: P-invariants (examples)




State machines
of components



# PetriDotNet: T-invariants (examples)

Cyclic behavior (here: correct, data loss)