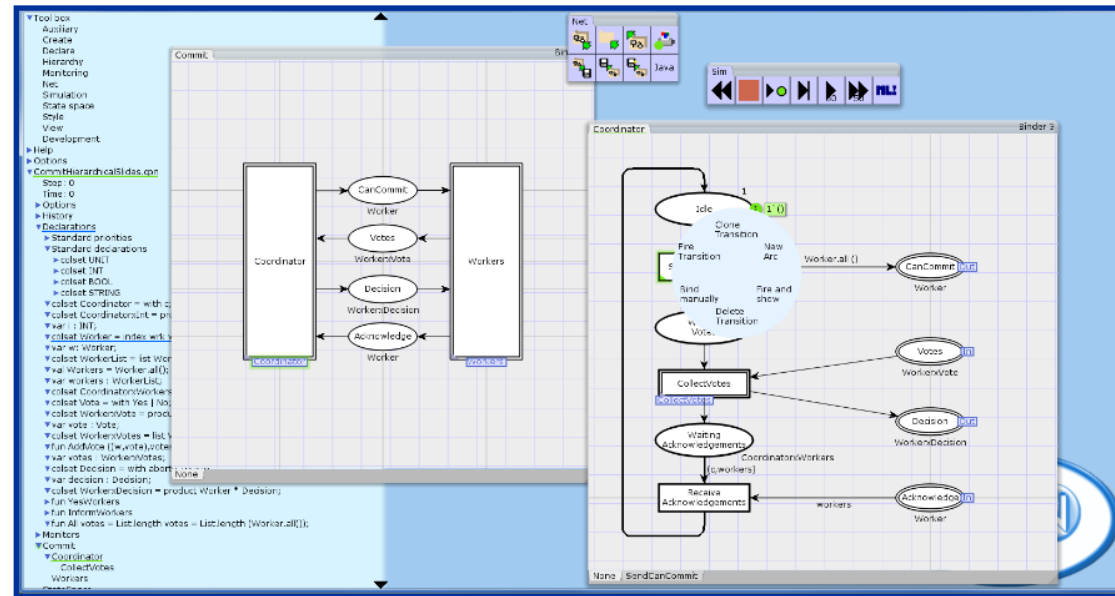


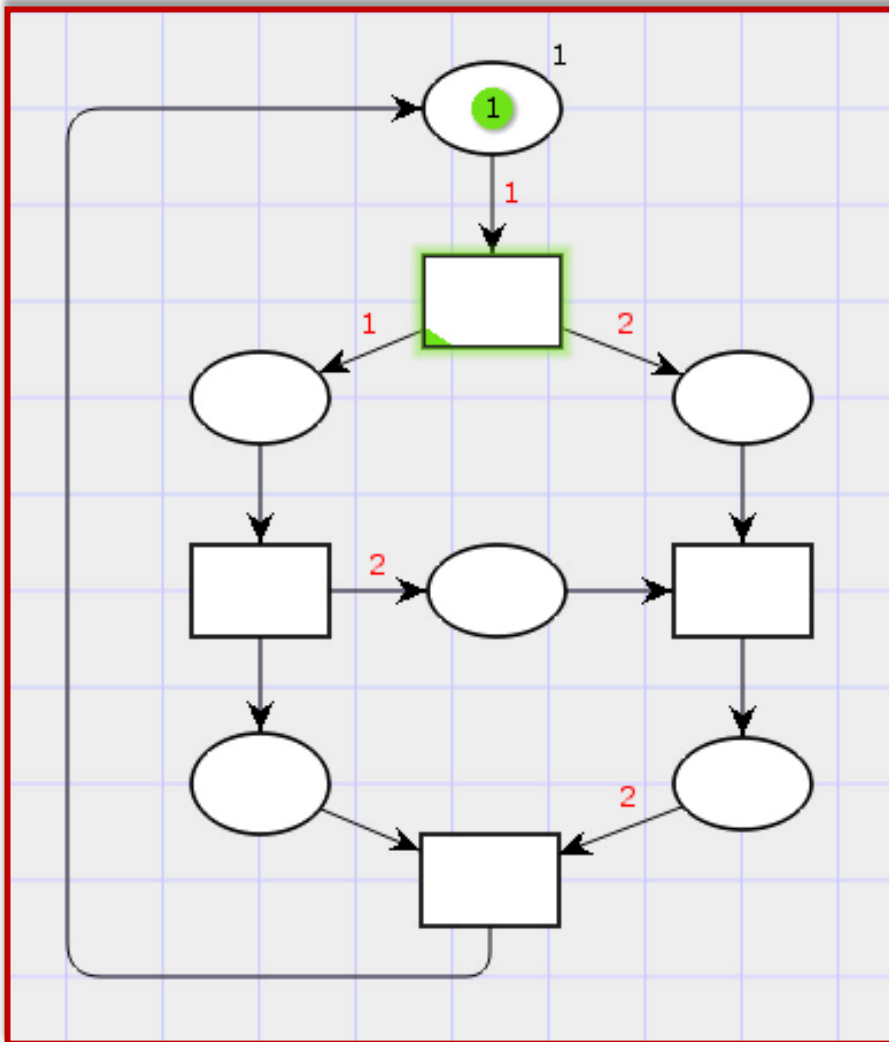
# Lecture 3

## Coloured Petri Nets



**Lars M. Kristensen**  
**Department of Computer Science, Electrical Engineering, and Mathematical Sciences**  
**Western Norway University of Applied Sciences**  
**Email: [lmkr@hvl.no](mailto:lmkr@hvl.no) / WWW: [home.hib.no/ansatte/lmkr](http://home.hib.no/ansatte/lmkr)**

# Quick Recap: Petri Net Concepts



## State modelling

- **Places** (ellipses) that may hold **tokens**
- **Marking (state)**: distribution of **tokens** on the places
- **Initial marking**: initial state

## Event (action) modelling

- **Transitions** (rectangles)
- **Directed arcs**: connecting places and transitions
- **Arc weights**: specifying tokens to be added/removed

## Execution (token game)

- **Current marking**
- **Transition enabling**
- **Transition occurrence**

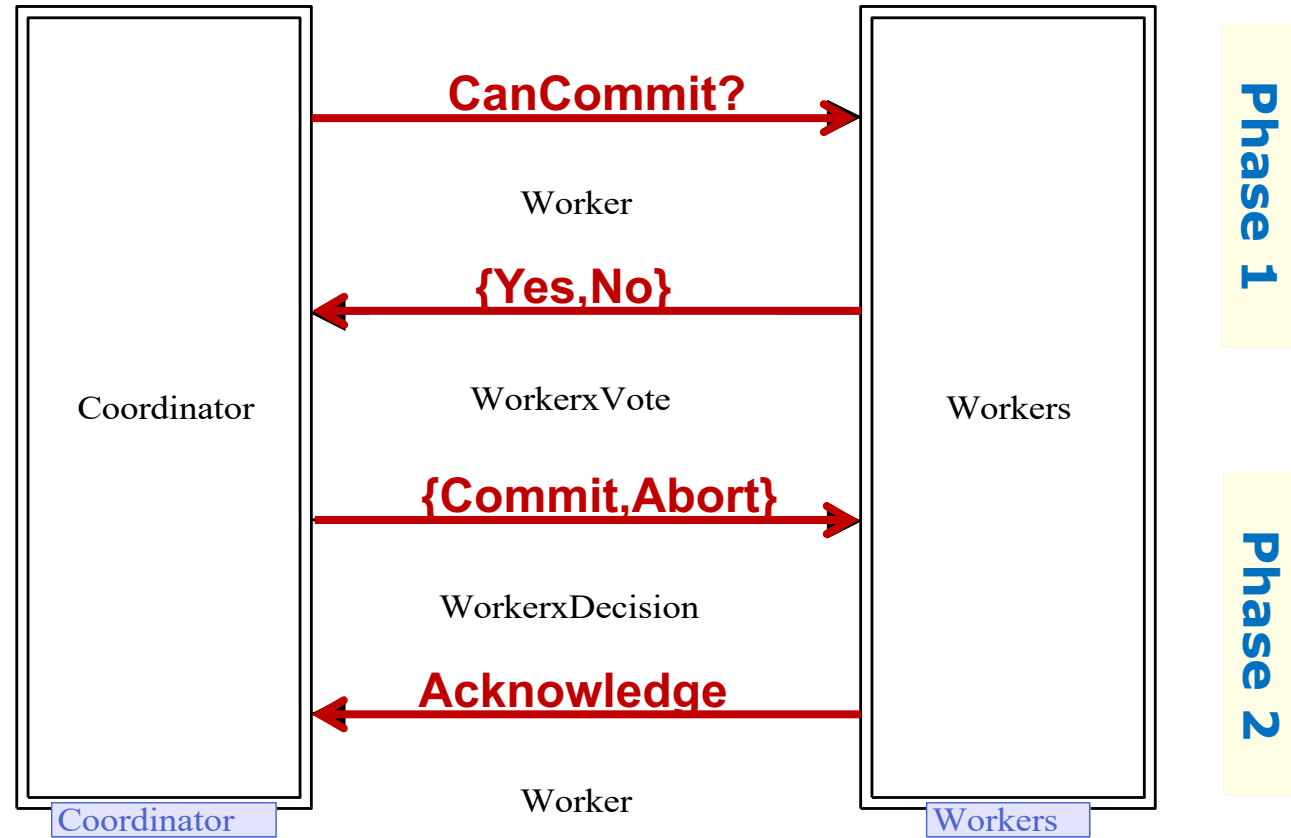
# Why do we need CPNs ?

- **CPNs include the basic syntactical and semantical concepts of Place/Transition Nets**
  - The black/anonymous PT-net tokens are represented using the UNIT type and the unit value ()
- **A main limitation of Place/Transitions Nets is scalability to large (real) concurrent systems**
  - Does not support parametric systems in an elegant way
  - Modelling of data is inconvenient
  - Does not allow models to be split into modules
- **CPNs also inherits additional language constructs**
  - Inhibitor arcs and reset arcs
  - Transition priorities

# CPN overview

- Address the practical shortcomings of PT-nets
- **Coloured Petri Nets (CPNs) = PT-nets + Standard ML programming language**
  - Places have a **type** (colour set) and tokens can carry **data values**
  - Transitions may have **variables** that can be **bound to values**
  - **Arc expressions** determine the tokens added/removed
  - **Guard expressions** may be used as an extra enabling condition
- **Standard ML = functional programming**
  - Computation proceeds by **evaluation of expressions**
  - **Static typing** with the type of expressions being **inferred**
  - **Functions** are first-order values and can be polymorphic
  - **Recursion** and lists are used to express iteration

# Two-phase Commit Transaction Protocol



# Colour set definitions

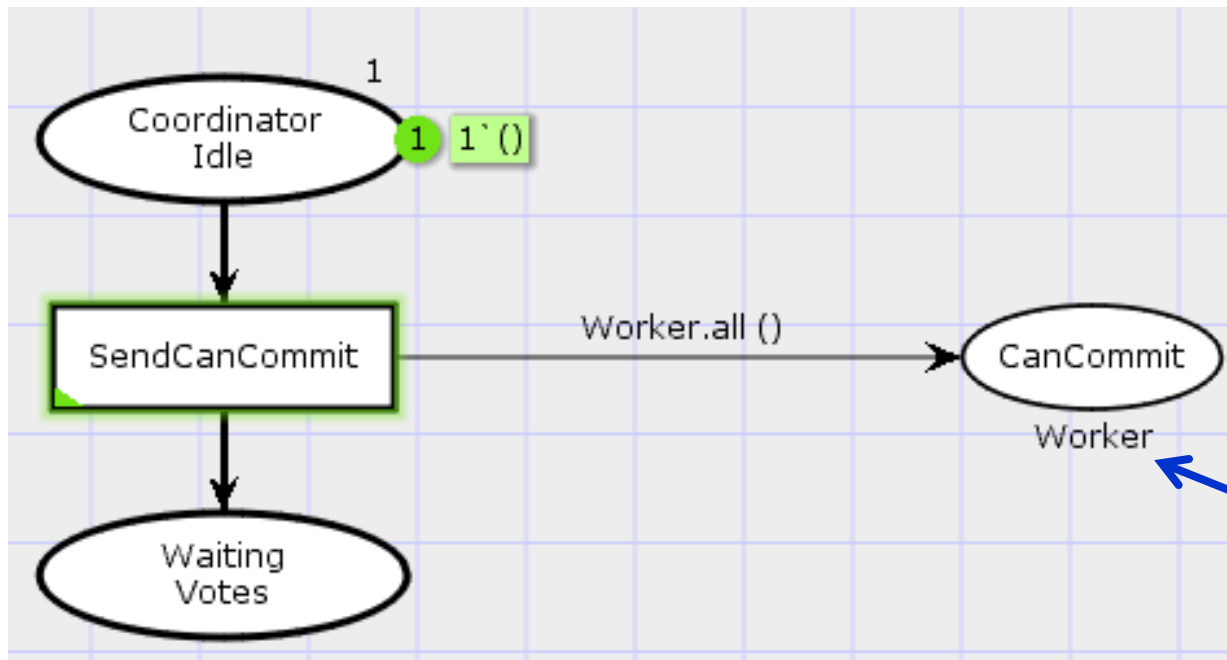
- Determines the **data types** that can be used in the model

Colour set definitions	Example values
<pre>val W = 2; colset Worker      = index wrk with 1..W; colset Vote        = with Yes   No; colset WorkerxVote = product Worker * Vote;</pre>	<pre>wrk(1), wrk(2) Yes, No (wrk(1), Yes)</pre>
<pre>colset Decision      = with Abort   Commit; colset WorkerxDecision = product Worker * Decision;</pre>	<pre>Abort, Commit (wrk(1), Commit)</pre>

- Additional colour set constructors for lists (**list**), records (**record**), and unions (**union**)
- **Base data types:** **UNIT**, **INT**, **STRING**, **BOOL**, **REAL**

# Coordinator – First phase

- The colour set (type) of a place determine the **kinds of tokens** that may reside on the place



CoordinatorIdle and WaitingVotes have an implicit **UNIT** colour set

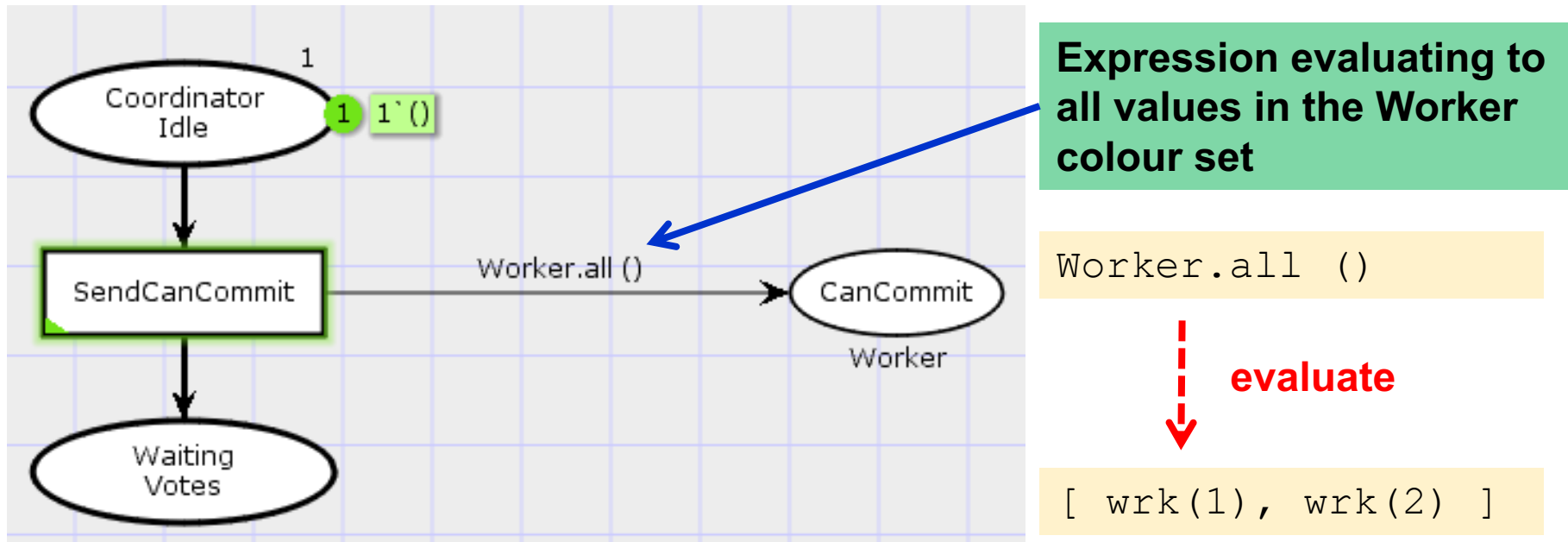
Tokens on CanCommit can have the values `wrk (1)` and `wrk (2)`

The colour set is by convention written below the place

**UNIT** is a datatype containing the single value `()` named unit

# Arc expressions

- Determine the tokens that are removed/added from/to places when transitions occur

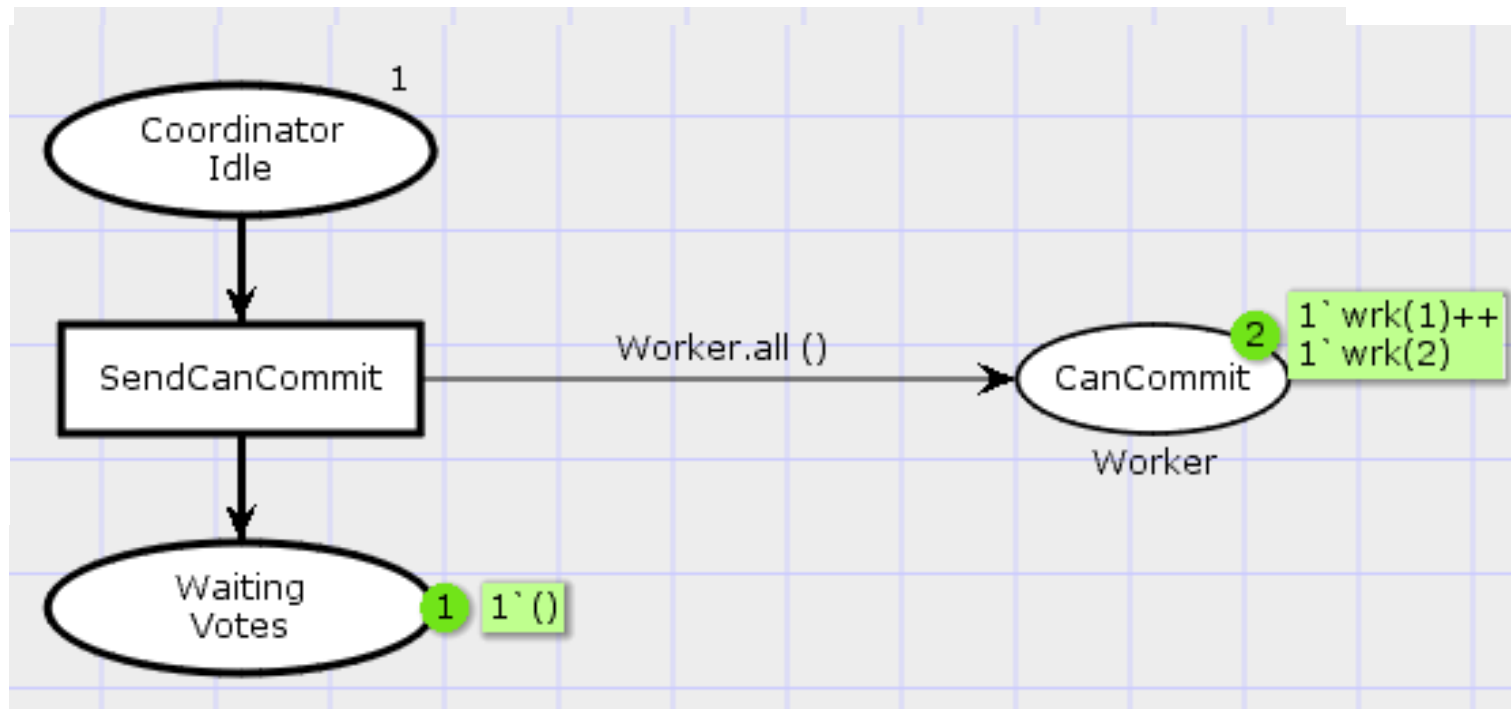


- The type of an arc expression **must match** the colour set of the place connected to the arc



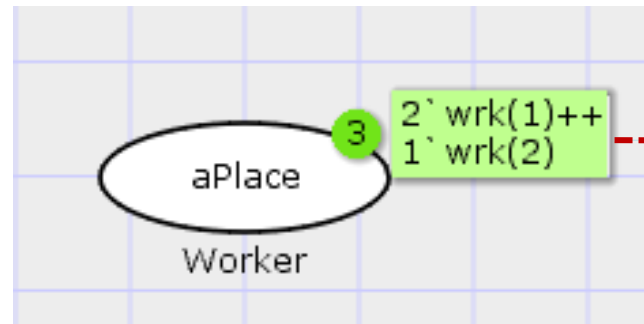
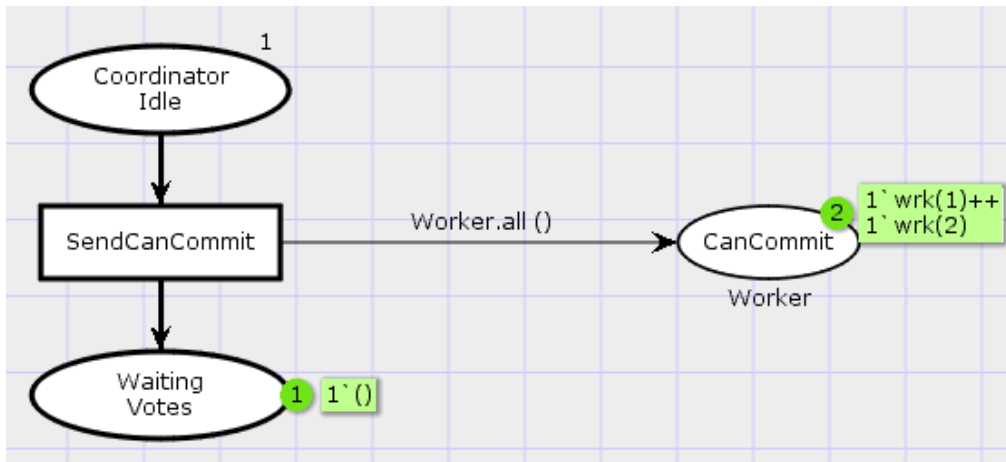
# Evaluation of expressions

- The tokens added and removed are determined by **evaluating arc expressions**



# Markings and multi-sets

- Each place may hold a **multi-set of tokens** over the colour set of the place



## Multi-set notation

coefficient («of»)

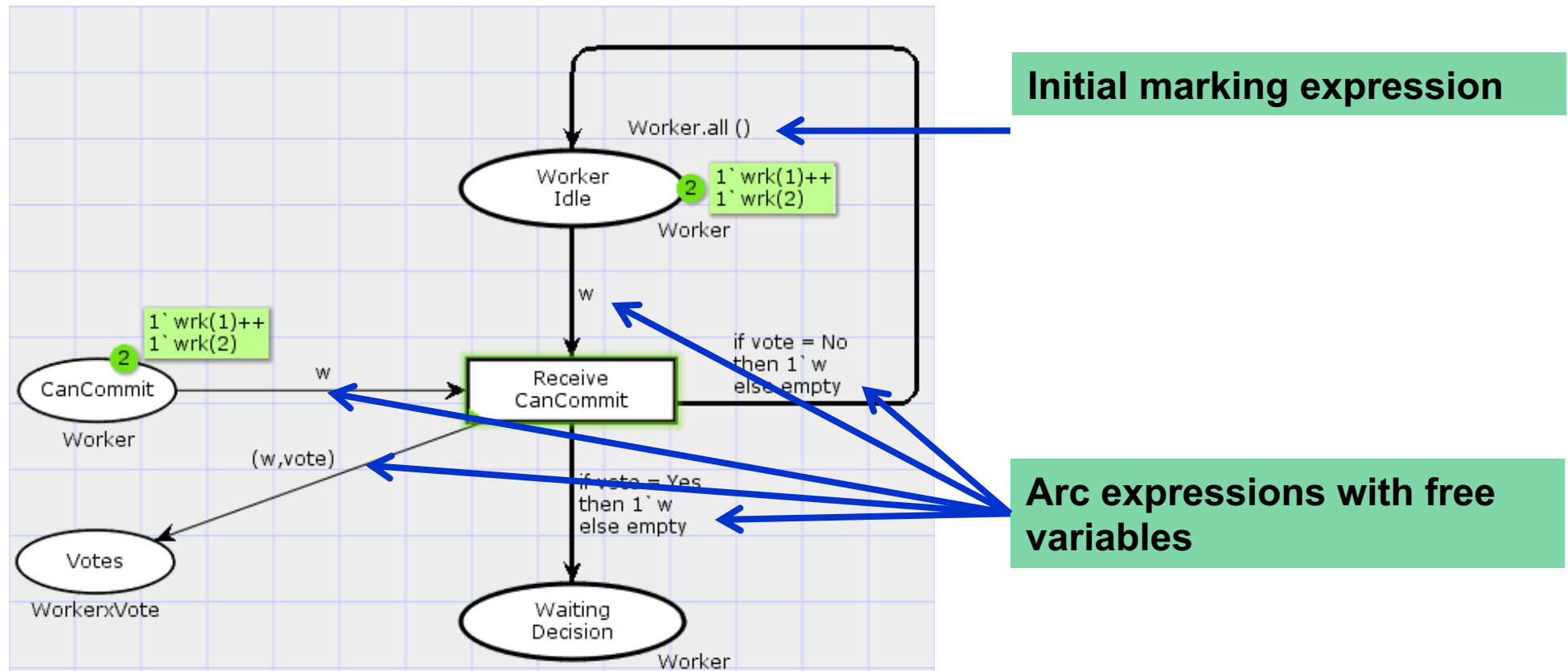
token colour  
(value)

2' wrk(1) ++  
1' wrk(2)

union («and»)

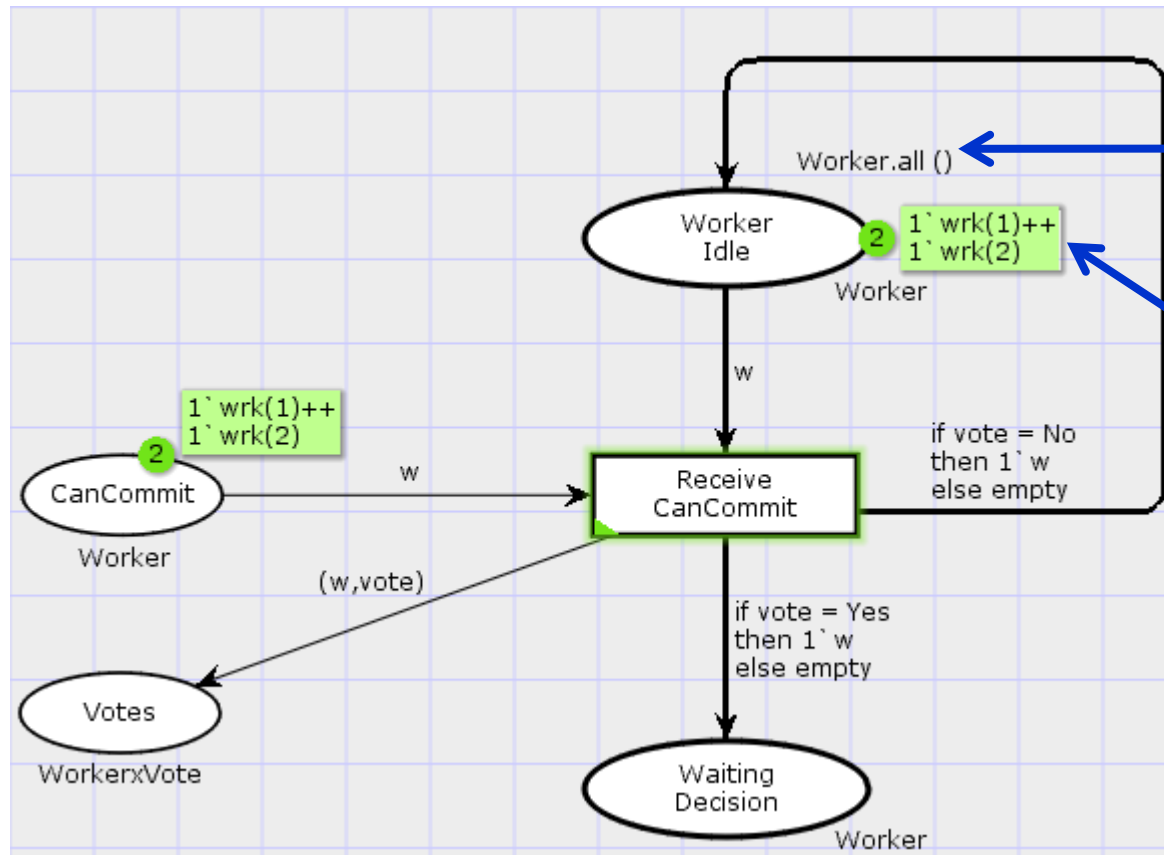
# Workers – First Phase

- Consists of receiving a CanCommit message and sending a decision (Yes/No) to the coordinator



# Initial marking

- The **initial marking** (state) is obtained by evaluating the **initial marking expressions**

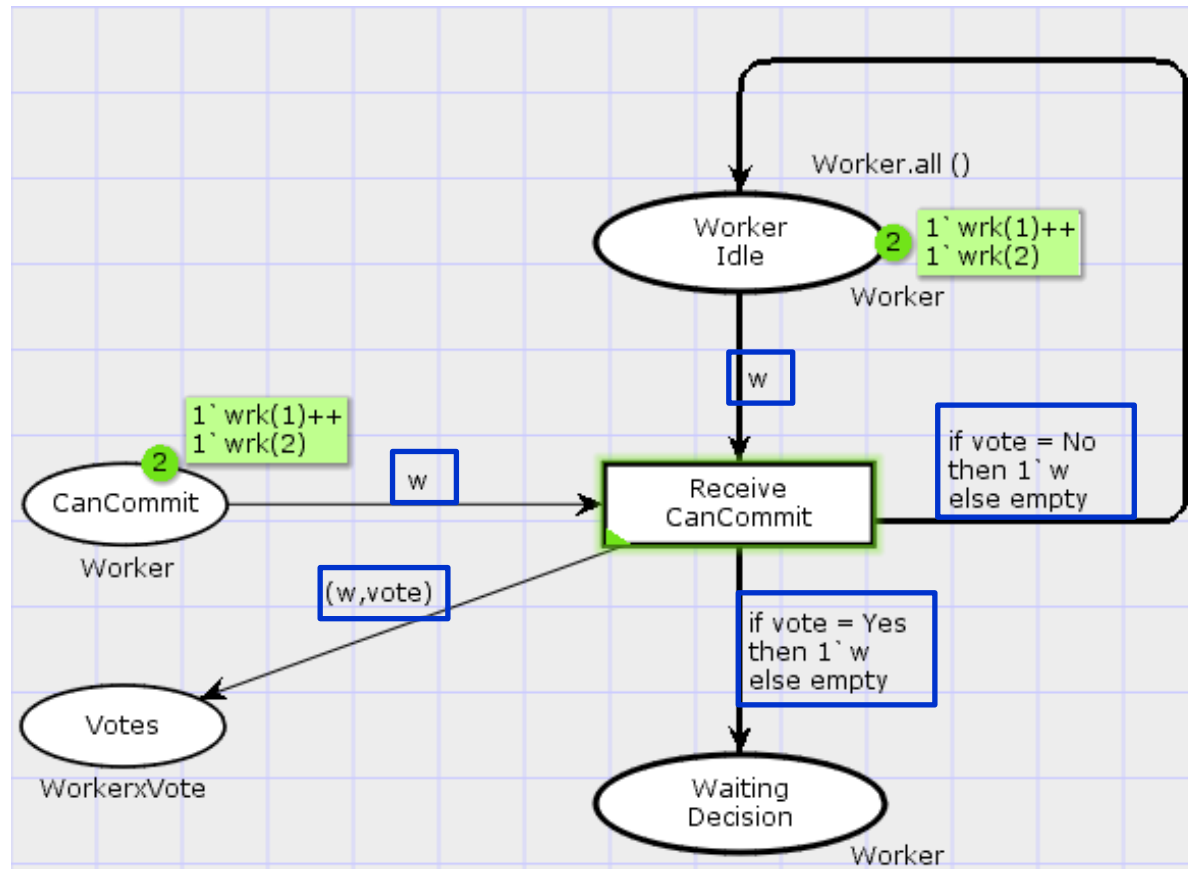


The initial marking is by convention written above the place

The two workers are initially **Idle**

# Transition variables

- The arc expressions on the arcs of a transition may contain **free variables**



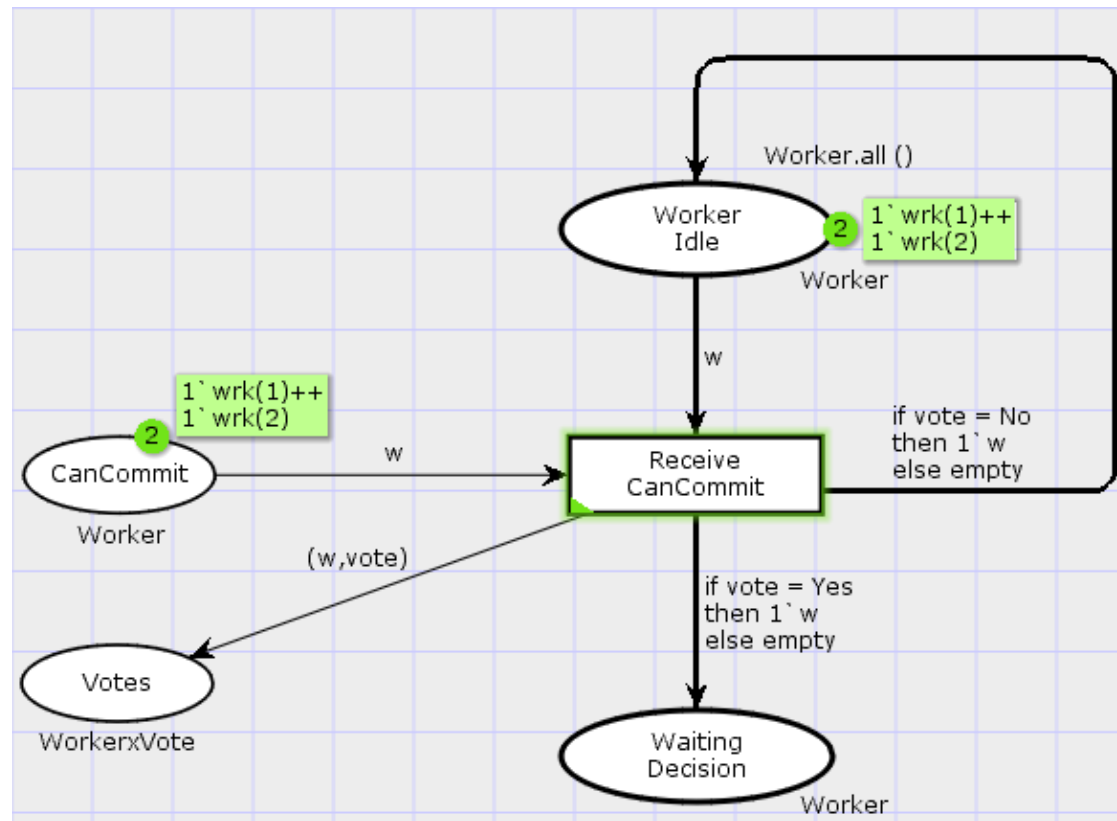
## Variable declarations

```
val W = 2;  
colset Worker =  
    index wrk with 1..W;  
var w : Worker;  
  
colset Vote = with Yes | No;  
var vote : Vote;
```

Arc expressions with free variables **vote** and **w**.

# Transition variables

- Transition **ReceiveCanCommit** has two free variables: **vote** and **w**



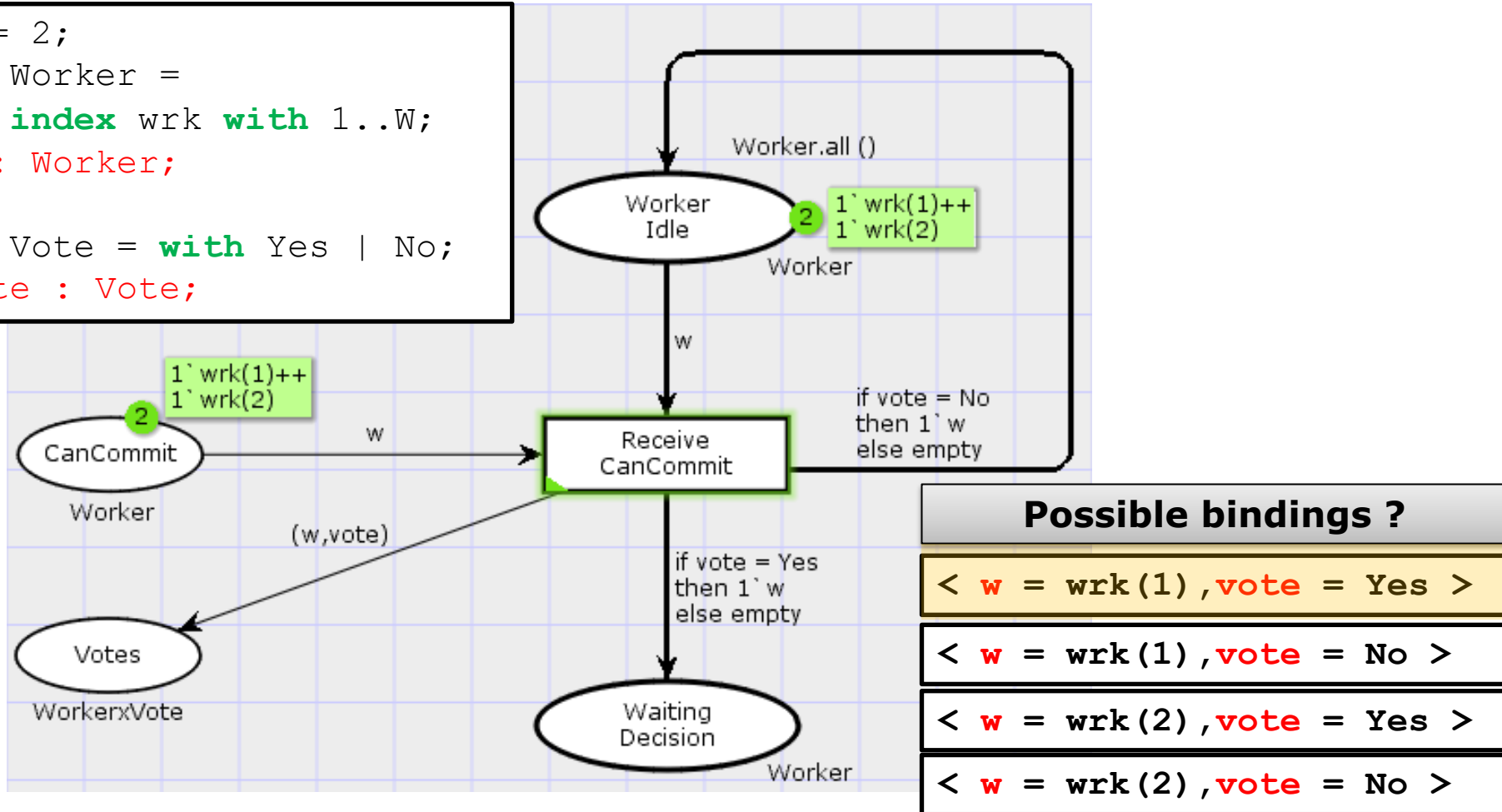
- Variables must be bound to values** for a transition to be enabled and occur
- Similar to formal and actual parameters known from programming
- The association of values to variables is called a **transition binding**
- The bindings correspond to the possible **enabling** and **occurrence modes** of the transition
- Not all possible bindings will in general be enabled

# Transition bindings

```

val W = 2;
colset Worker =
    index wrk with 1..W;
var w : Worker;

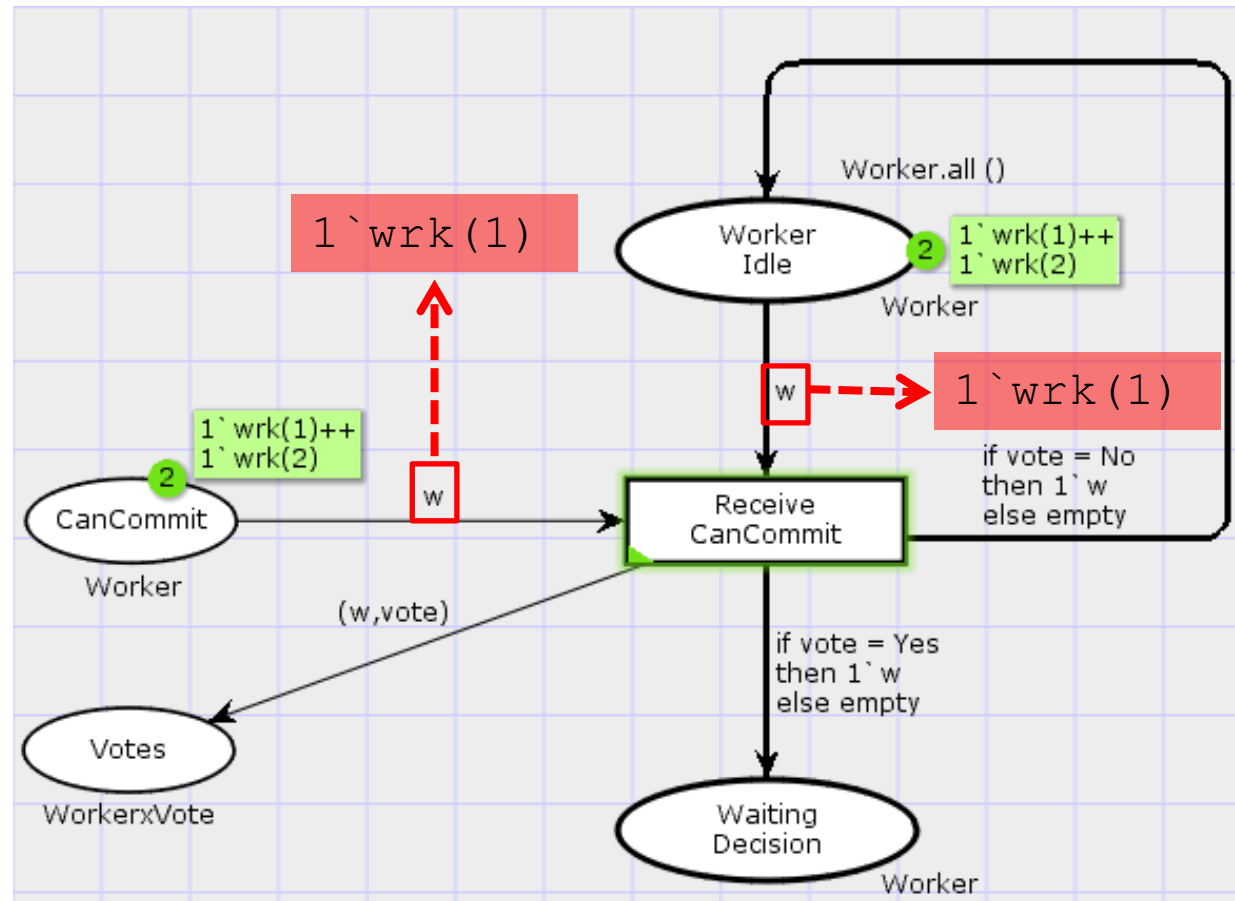
colset Vote = with Yes | No;
var vote : Vote;
    
```



- The **scope of a variable** is the surrounding arc expressions of the transition

# Transition binding enabling

**< w = wrk(1), vote = Yes >**

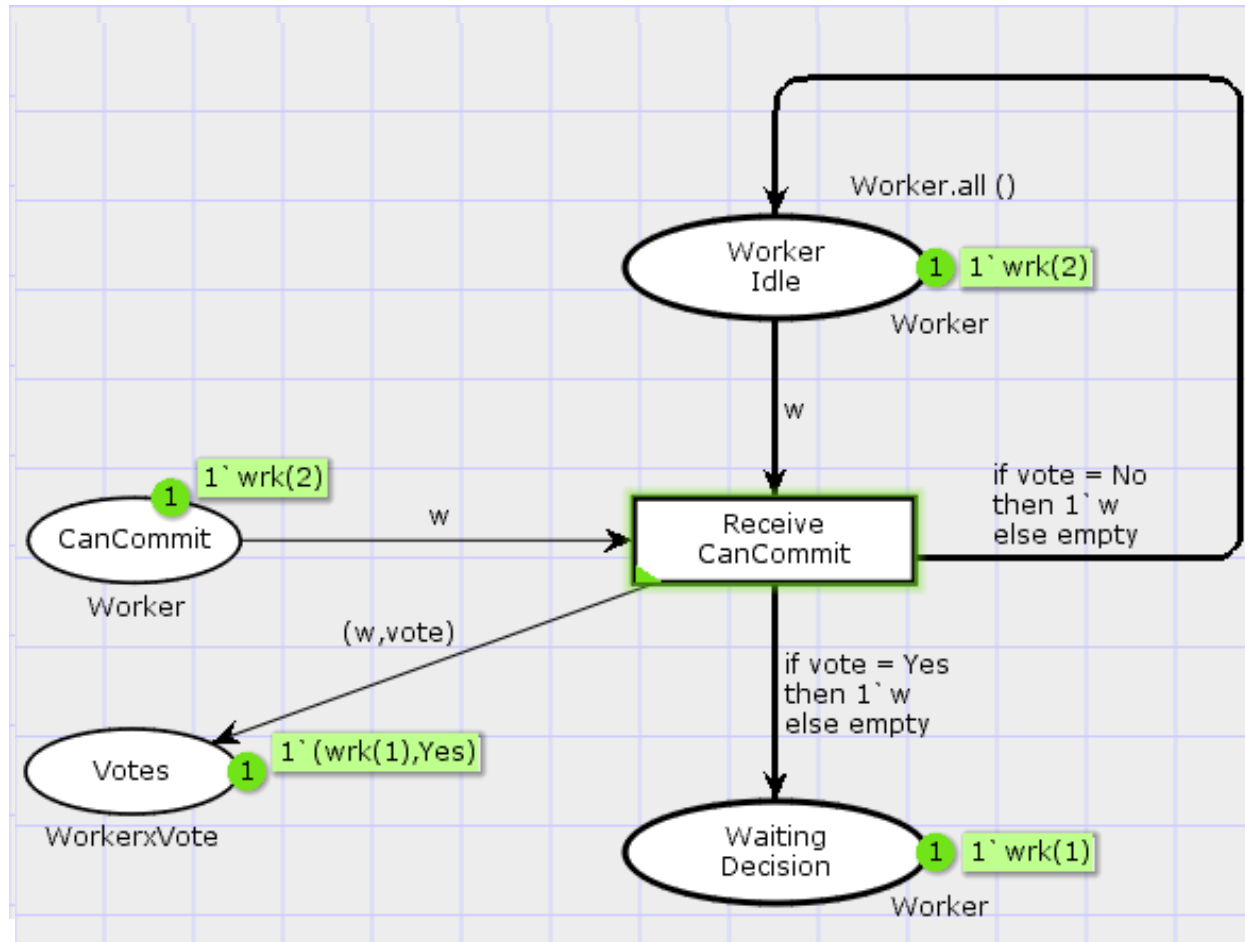


- A transition binding is **enabled** if there are sufficient tokens on each input place
- **Tokens required on input places** are determined **by evaluating the input arc expressions in the binding** under consideration
- **Enabling condition:** the multi-set of tokens obtained must be **contained in** the multi-set of tokens present on the corresponding input place



# Transition binding occurrence

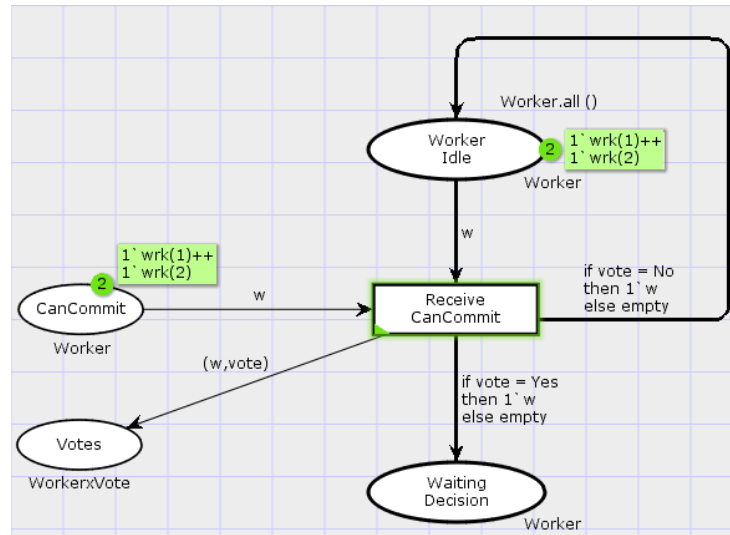
**< w = wrk(1), vote = Yes >**



- An enabled transition binding may **occur** changing the current marking (state)
- **Tokens removed from input places:** determined by evaluating the input arc expression in the binding
- **Tokens added to output places:** determined by evaluating the output arc expressions in the binding

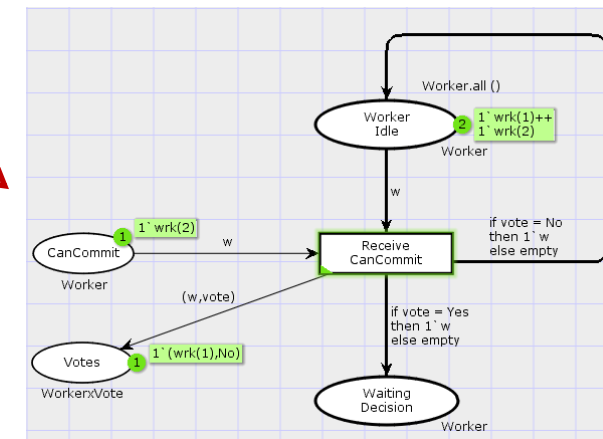
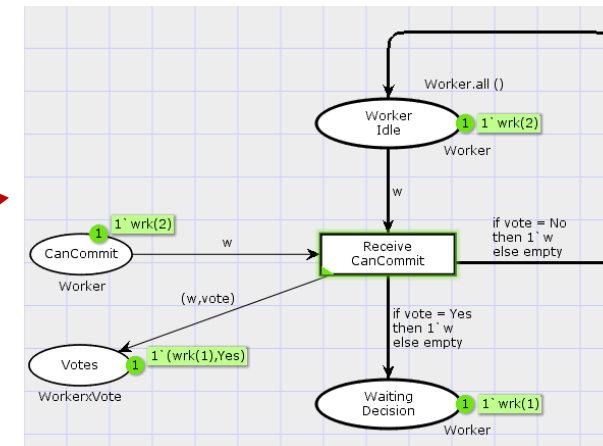
# Binding occurrence

- A transition may have several enabled bindings



$b_{1Y}$

$b_{1N}$



## Bindings

$b_{1Y} = \langle w = wrk(1), vote = Yes \rangle$

$b_{1N} = \langle w = wrk(1), vote = No \rangle$

$b_{2Y} = \langle w = wrk(2), vote = Yes \rangle$

$b_{2N} = \langle w = wrk(2), vote = No \rangle$

# CPN Tools demo

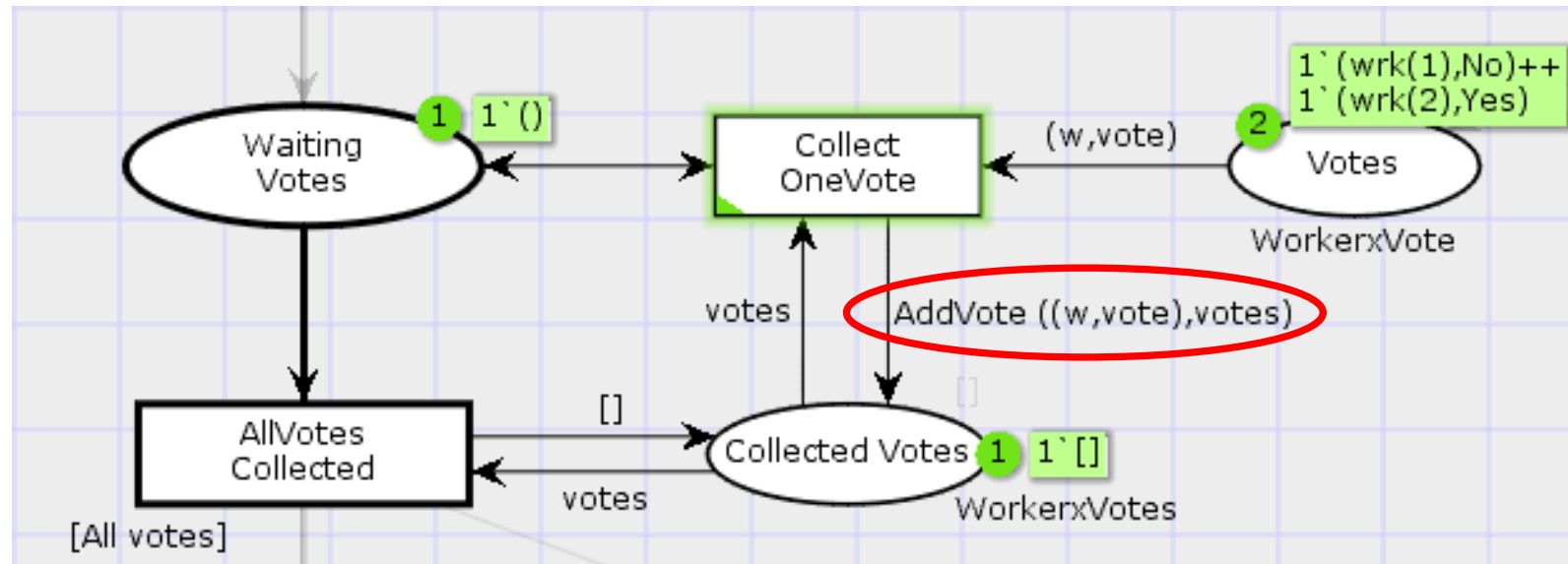
- **Simulation of CPN models**
  - Interactive simulation with binding selection
  - Returning to the initial marking
  - Automatic simulation with visual feedback
  - Stop options and automatic simulation



<https://github.com/lmkr/cpncourse/blob/master/models/lecture3-cpns.cpn>

# Collecting votes

- Votes are collected one at a time and accumulated in a list-token on place **Votes**



```

var w      : Worker;      var vote : Vote;
colset WorkerxVote = product Worker * Vote;
colset WorkerxVotes = list WorkerxVote;

var votes : WorkerxVotes;
    
```

# Functions

- The function **AddVote** is used to add a vote from a worker to the list of collected votes

```
fun AddVote ((w,vote), votes) = (w,vote) :: votes
```

- **Example**

```
AddVote ((wrk(1), No), [])
```



```
(wrk(1), No) :: []
```



```
[(wrk(1), No)]
```

```
AddVote ((wrk(2), Yes), [(wrk(1), No)])
```



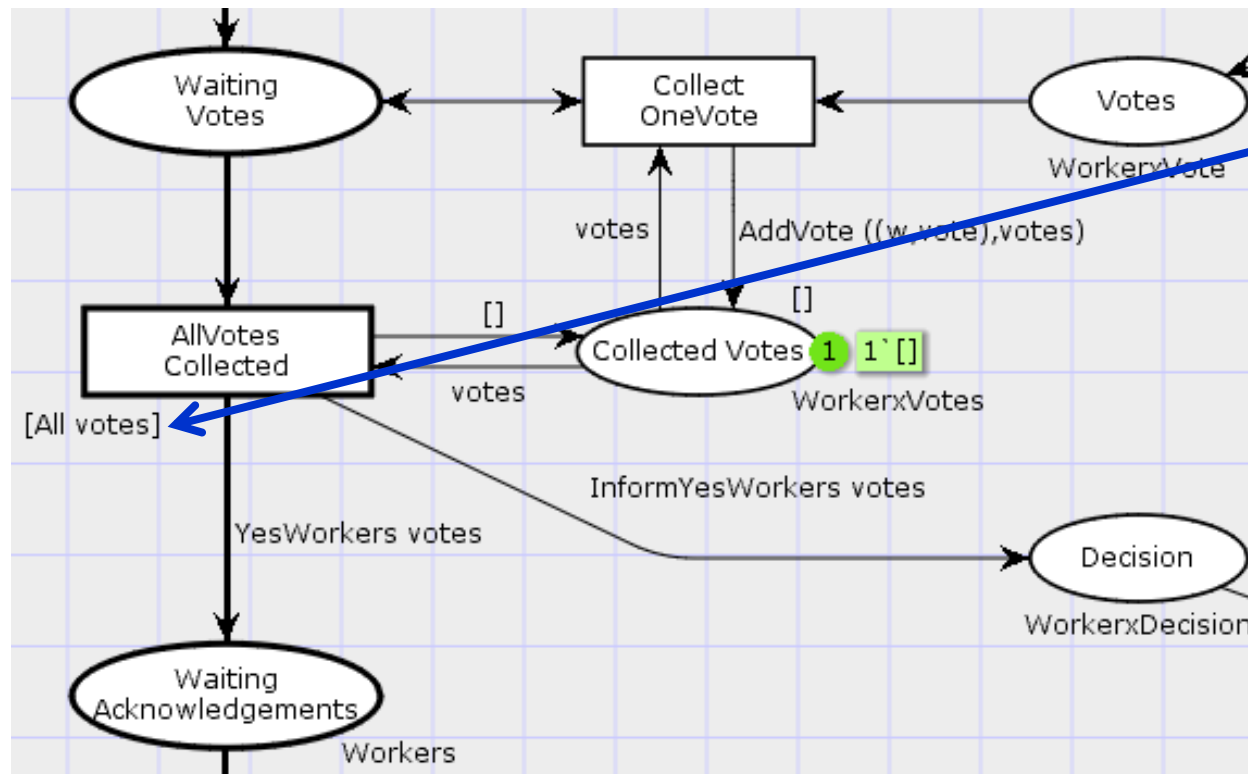
```
[(wrk(2), Yes), (wrk(1), No)]
```

**:: is the list-constructor  
in Standard ML**



# Guard expressions

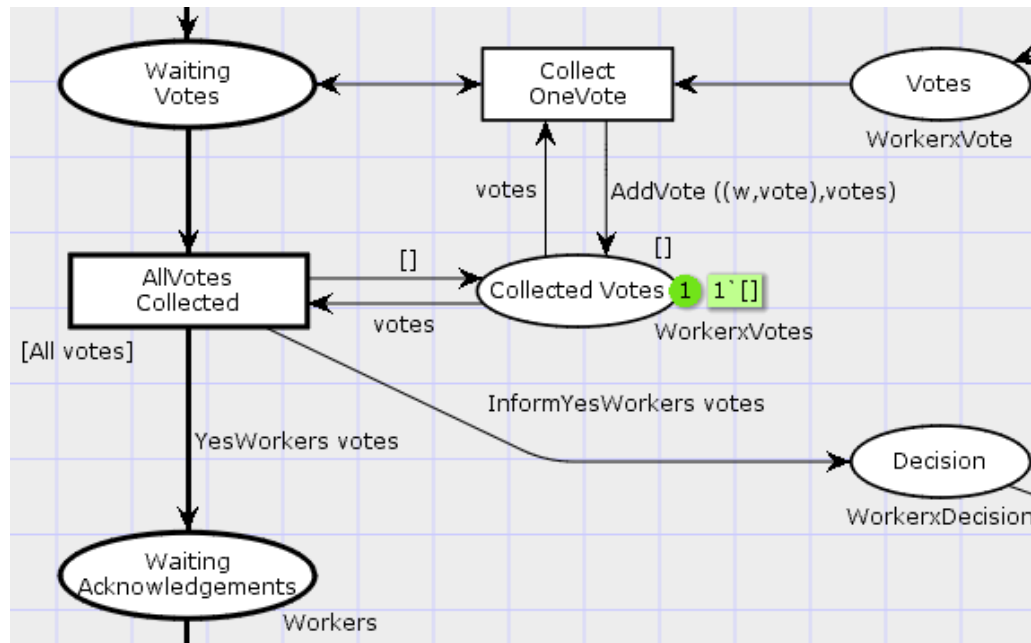
- A transition may have a boolean **guard expression** which is extra enabling condition



**The guard is by convention written in square brackets next to the transition**

# Collecting all votes

- Transition **AllVotesCollected** should only be enabled when we have collected all votes



```
colset WorkerxVote =  
    product Worker * Vote;
```

```
colset WorkerxVotes =  
    list WorkerxVote;
```

```
var votes : WorkerxVotes
```

```
fun All votes =  
    (List.length votes = W)
```

```
votes = [(wrk(2), Yes), (wrk(1), No)]
```

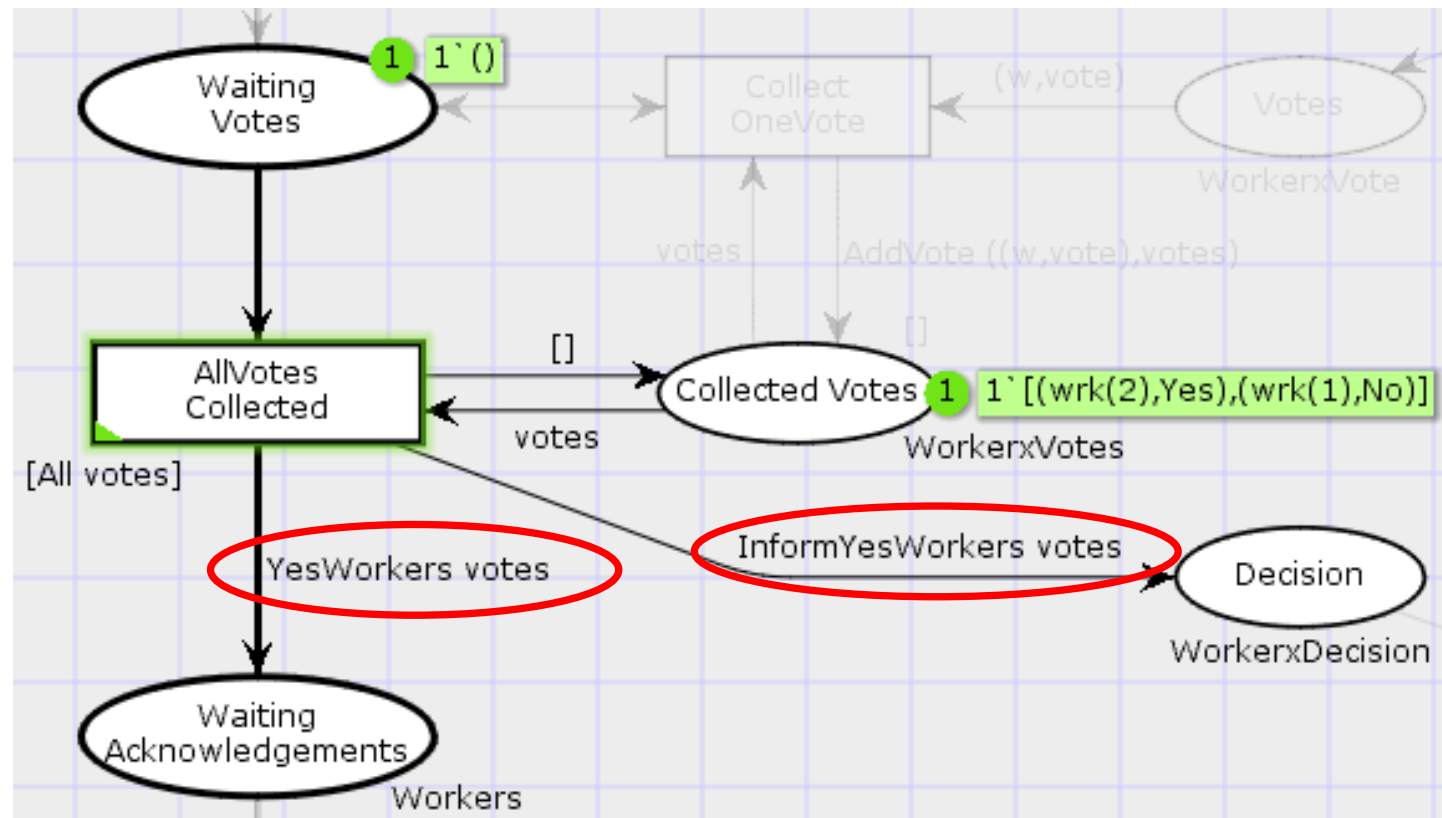
```
All votes ----> true
```

```
votes = [(wrk(1), No)]
```

```
All votes ----> false
```

# Informing workers

- Functions are also used to find the workers that needs to be informed about the decision





# YesWorkers Functions

- **Obtaining the list of Yes-votes (utility function)**

```
fun yesVotes votes = List.filter  
    (fn (w,vote) => vote = Yes)  
    votes
```

```
yesVotes [(wrk(1),No), (wrk(2),Yes)]
```



```
[(wrk(2),Yes)]
```

- **Getting workers that votes Yes (projection)**

```
fun YesWorkers votes = List.map  
    (fn (w,_) => w)  
    (yesVotes votes)
```

# InformYesWorkers Functions

```
fun InformYesWorkers votes =  
  let  
    val yesworkers = YesWorkers votes  
    val decision =  
      (if (List.length yesworkers = W)  
        then commit  
        else abort)  
  in  
    List.map (fn w => (w, decision)) yesworkers  
end
```

```
InformYesWorkers [(wrk(2), Yes)]
```

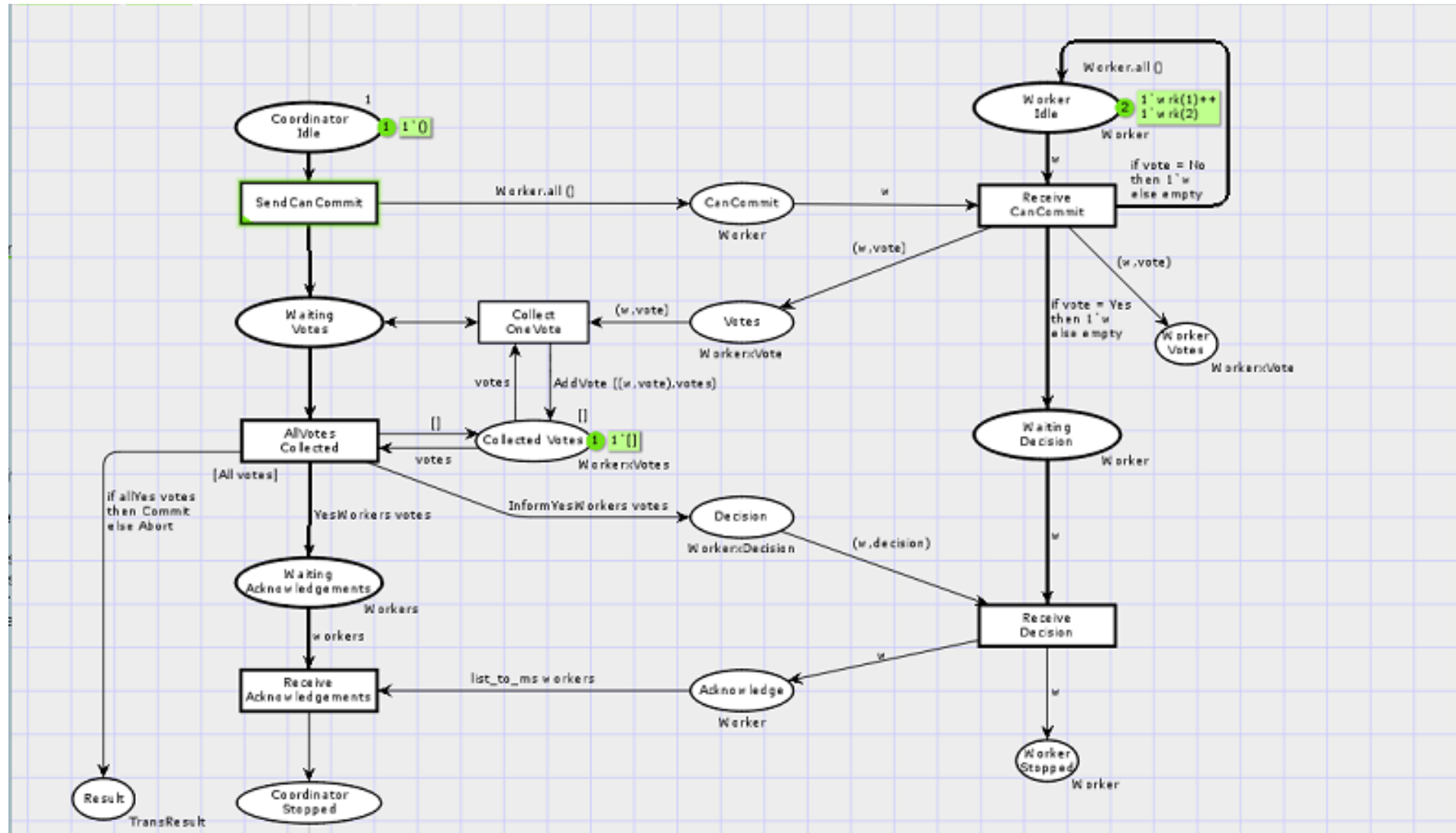


```
[(wrk(2), abort)]
```

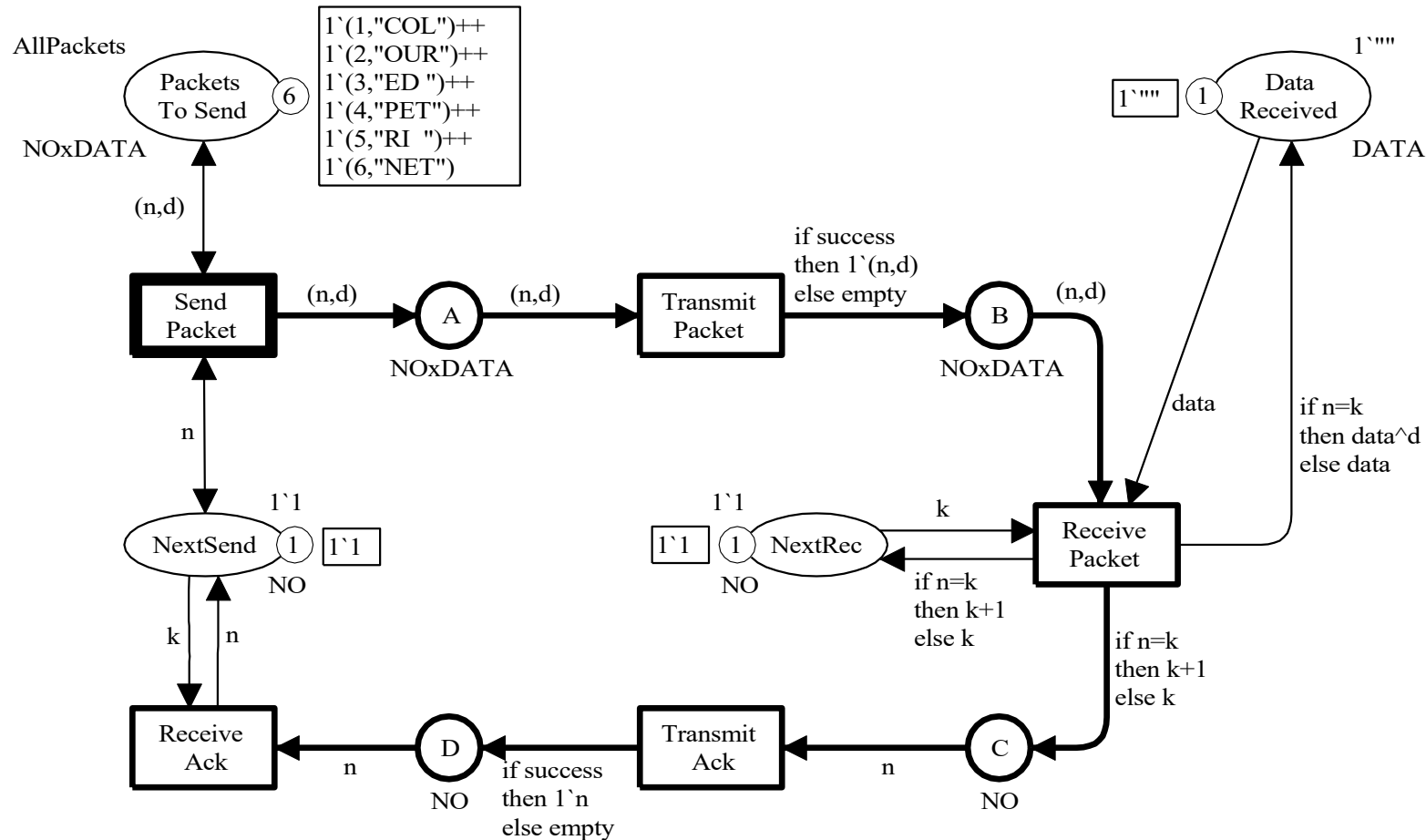
# CPN Tools demo

- **Editing of CPN models**
  - Incremental syntax check of the model (dependencies)
  - Adding and deleting declarations
  - Editing inscriptions  
(arc expressions, colour sets, initial markings, guards)
  - Guidelines, graphical attributes, and groups





# Second example – stop-wait protocol



<https://github.com/lmkr/cpnbook/blob/master/models/chapter2/2-10NondeterministicProtocol.cpn>

# Summary

- **Coloured Petri Nets extends Place/Transition Nets with a functional programming language**
- **Syntactical concepts**
  - **Colour sets** defines the data types available for modelling
  - Declaration of **variables** over the colour sets of the model
  - Places have a colour set determining the **kind of tokens** a place may contain
  - Arc expressions, initial marking- and guard expressions
- **Semantical concepts**
  - The marking of a place is a **multi-set of tokens (values)**
  - A **binding** gives values to the variables of a transition – scope of a variable is the surrounding arc expressions of the transition
  - **Evaluation of arc expressions and guards** in bindings determine enabling and the tokens removed/added by transitions