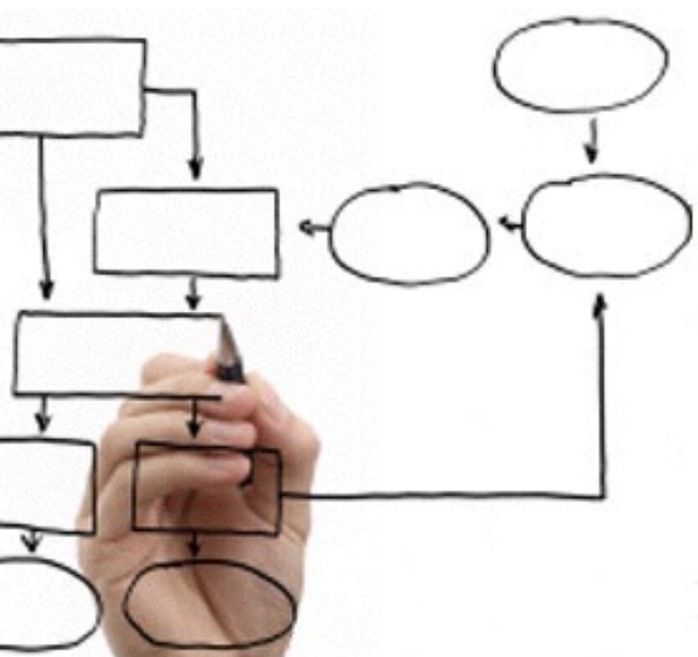
# Methods for the specification and verification of business processes

MPB (6 cfu, 295AA)



Roberto Bruni

http://www.di.unipi.it/~bruni

## 19 - Diagnosis for WF nets

# Object

We study suitable diagnosis techniques for unsound Workflow nets

Diagnosing workflow processes using Woflan (article, optional reading)
http://wwwis.win.tue.nl/~wvdaalst/publications/p135.pdf

# Some Pragmatic Considerations

We know that, for free-choice nets,
liveness and boundedness can be decided efficiently
(in polynomial time)

but we want to check soundness for a wider range of nets

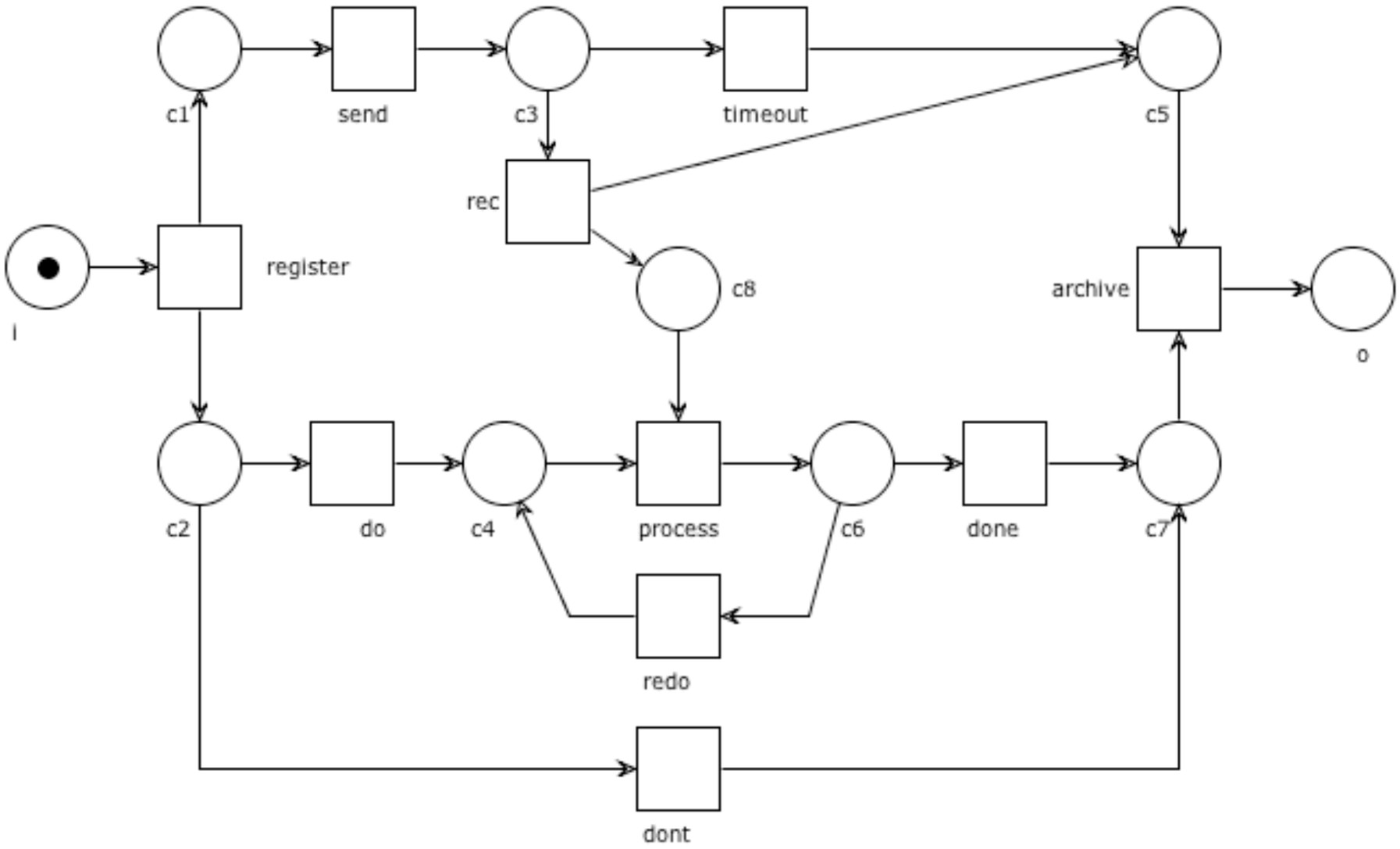
Moreover, when a process is not sound, some diagnostic can be generated that indicates why it is flawed

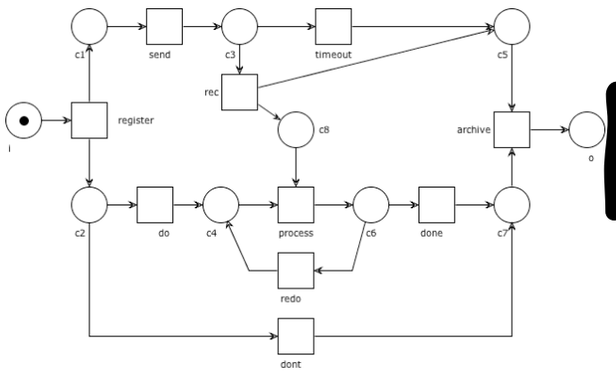
# Woflan
# (now a ProM plugin)

WOrkFLow ANalyzer (Windows only)
http://www.win.tue.nl/woflan/

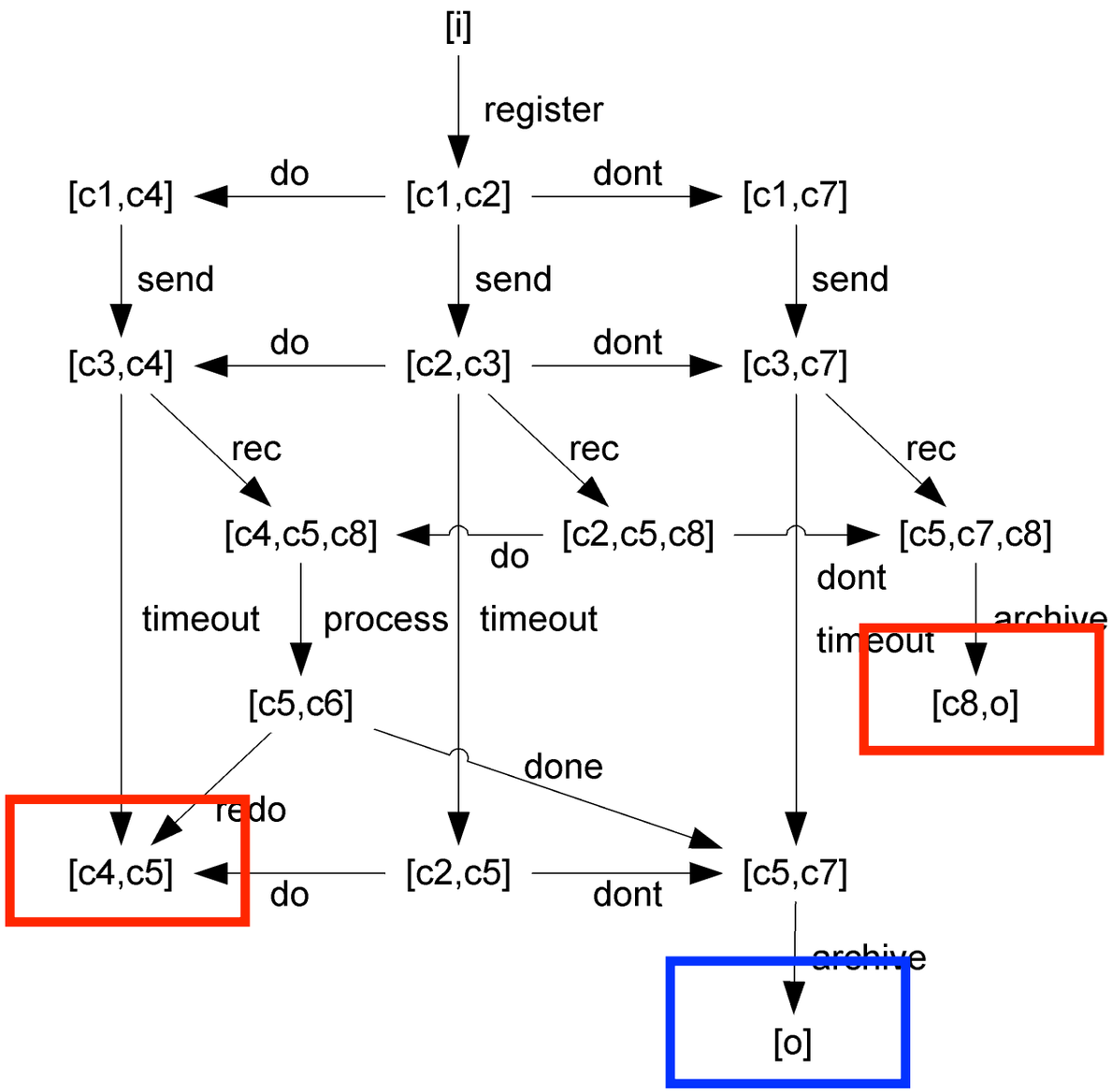
Woflan tells us if N is a sound workflow net
(Is N a workflow net?  Is N* bounded?  Is N* live?)
if not, provides some diagnostic information

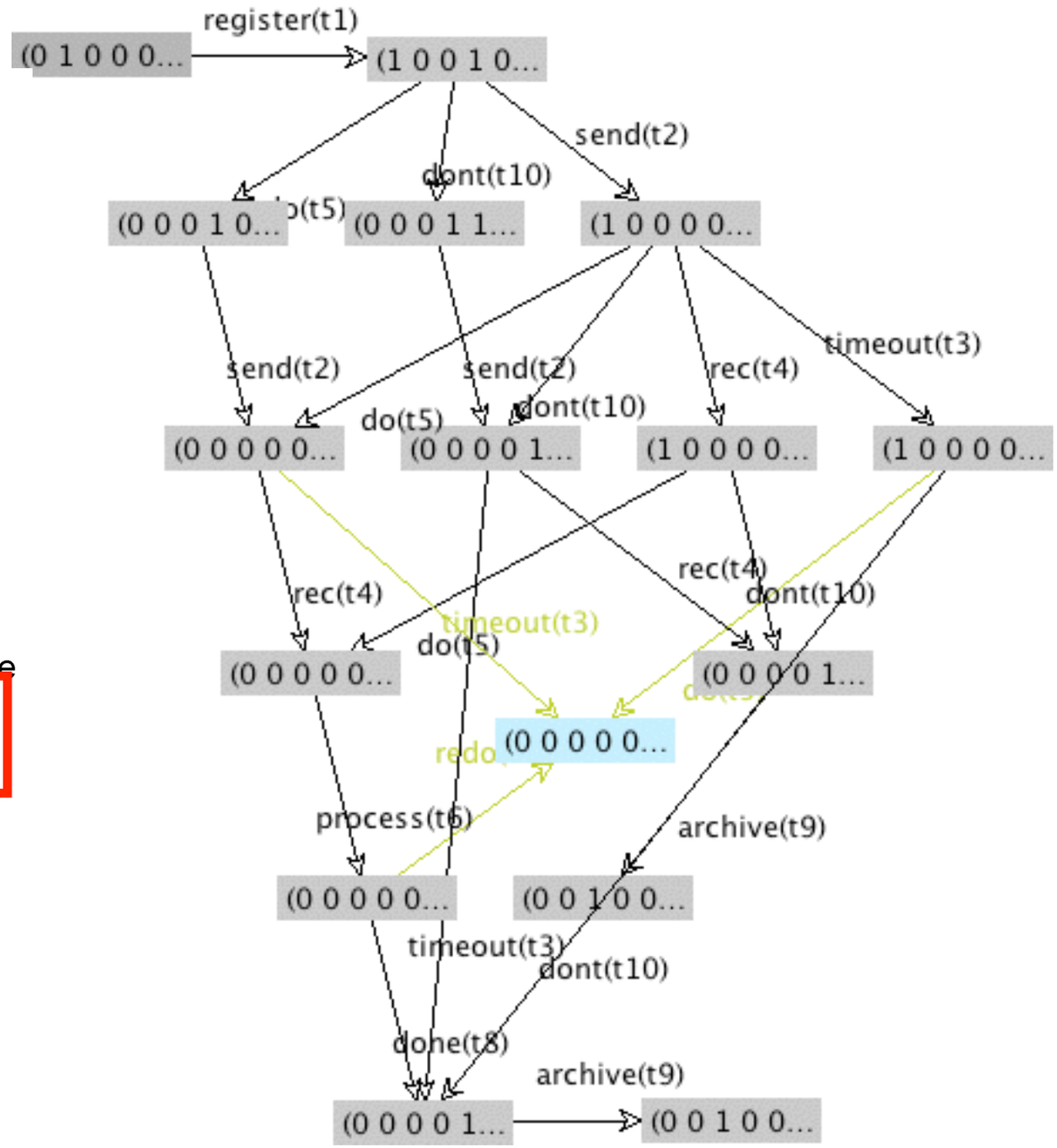
# Running example



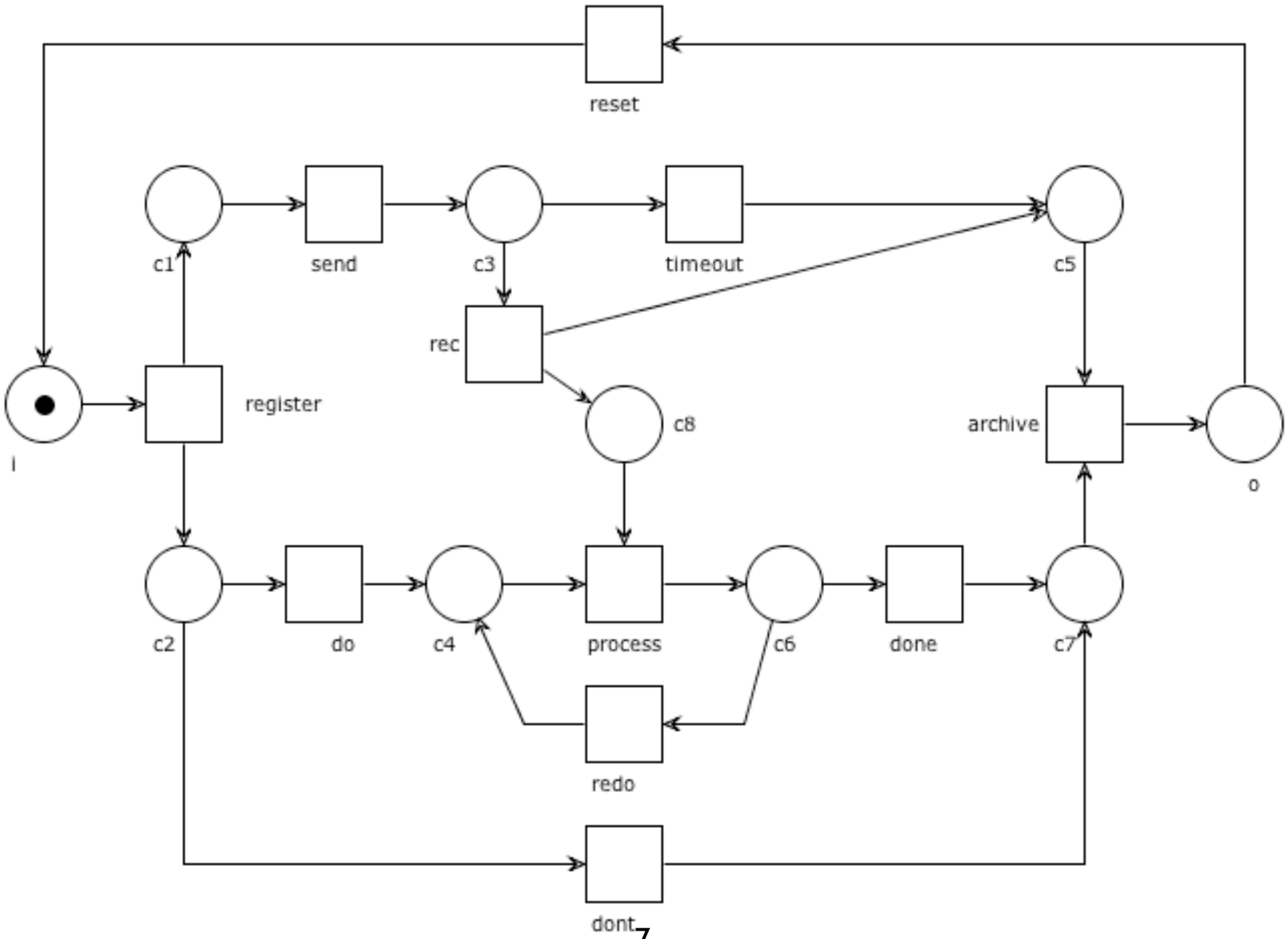


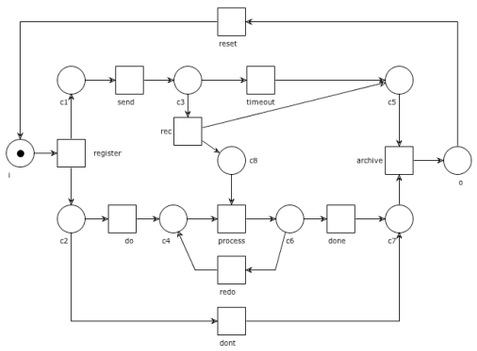

# Running example



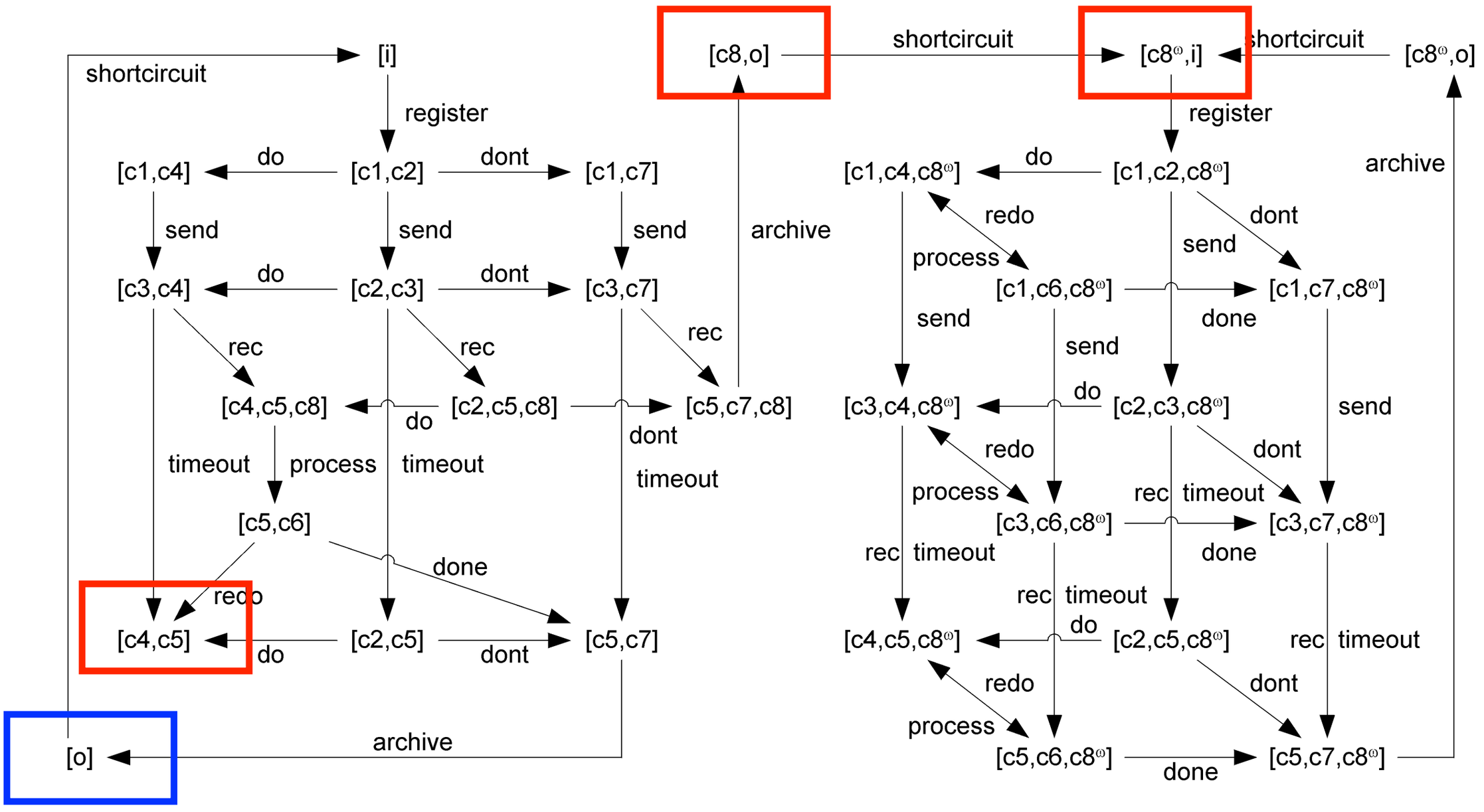



# Running example: short-circuited






# Running example: short-circuited



# Structural analysis

# Quick reminder

A **subnet** $N' = (P', T', F')$ of $N = (P, T, F)$ consists of:

- a subset $P' \subseteq P$ of places
- a subset $T' \subseteq T$ of transitions
- the subset $F \cap ((P' \times T') \cup (T' \times P')) \subseteq F$ of arcs

An **S-component** is a subnet $N' = (P', T', F')$ of $N$ that:

- is a strongly-connected S-net $(\forall t \in T'. |\bullet t| = |t \bullet| = 1)$
- for any $p \in P'$ we have $\bullet p \cup p\bullet \subseteq T'$

# Quick reminder

In a S-component,
the total number of tokens in its places is constant

Any S-component
induces a uniform invariant (weights 0 and 1)

A net is **S-coverable** iff
any $p \in P$ belongs to some S-component

S-coverability implies boundedness
(because it induces a positive S-invariant)

# S-Invariant analysis

If every place of N* is covered by a semi-positive S-invariant then N* is bounded

**Places not covered by semi-positive S-invariants are potential sources of errors**

# S-Coverability vs Soundness

S-coverability is one of the basic requirements any workflow process definition should satisfy

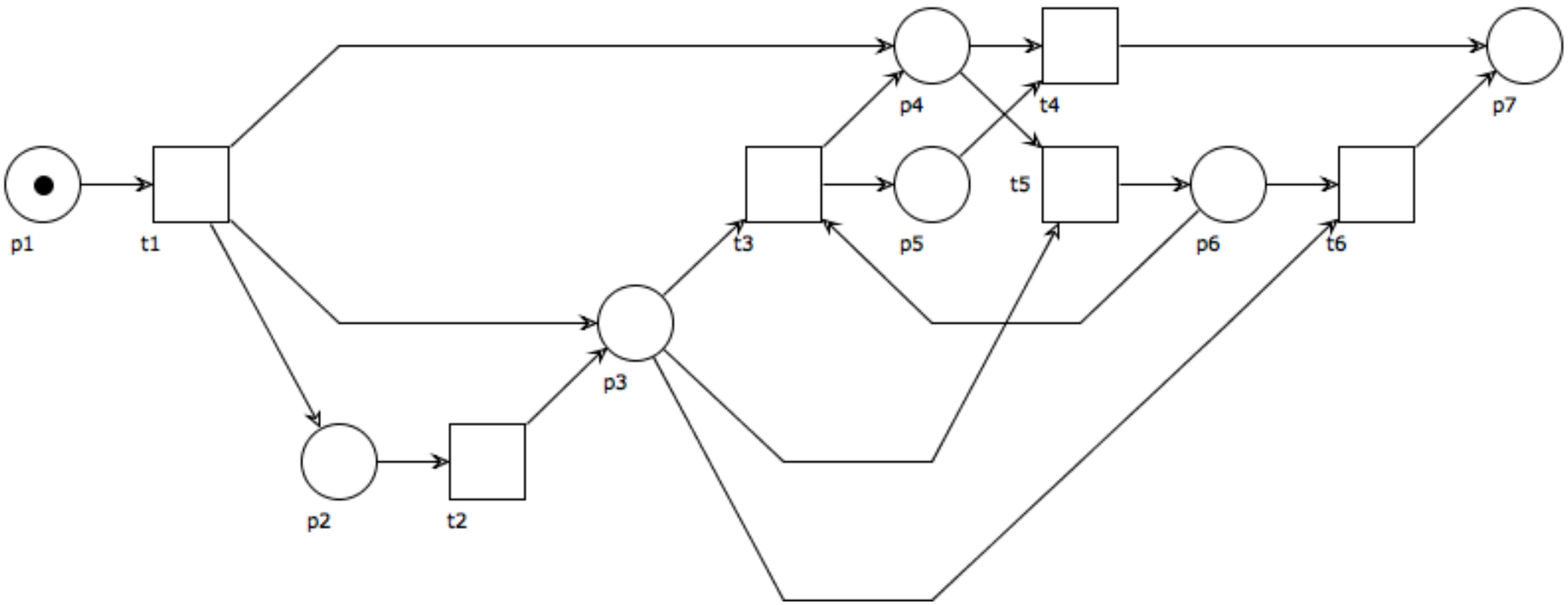
Still:

S-coverability is not a sufficient requirement for soundness

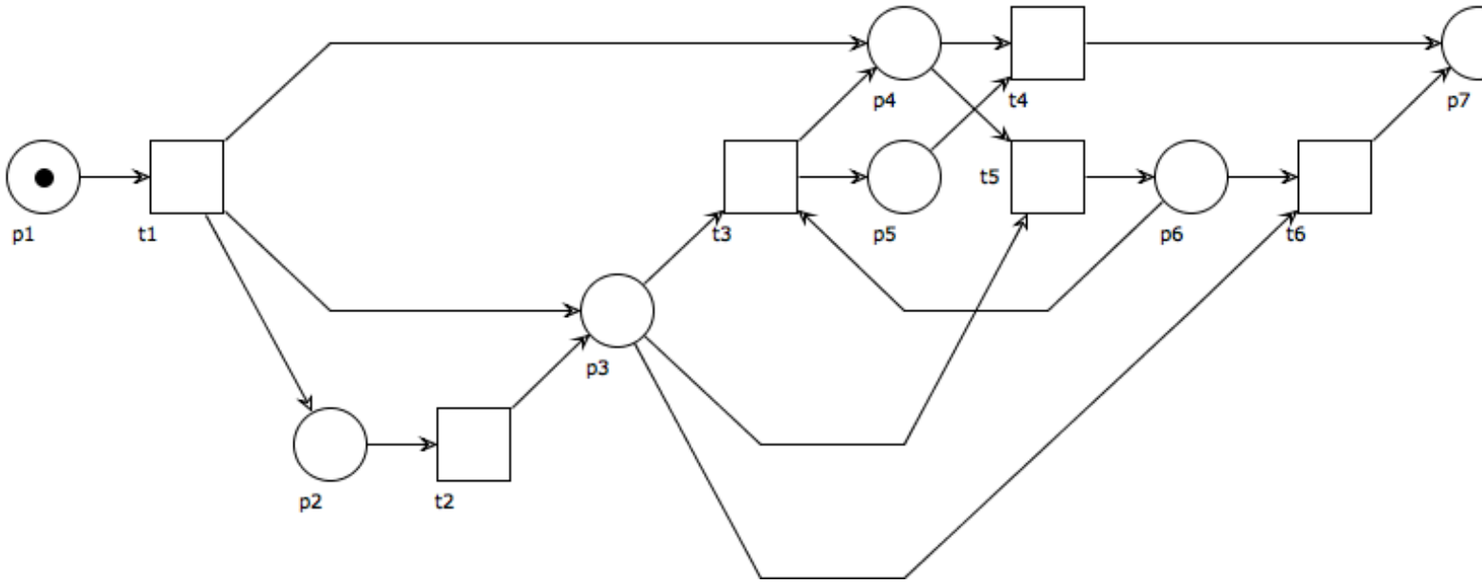
**N* can be S-coverable even if N is not sound**

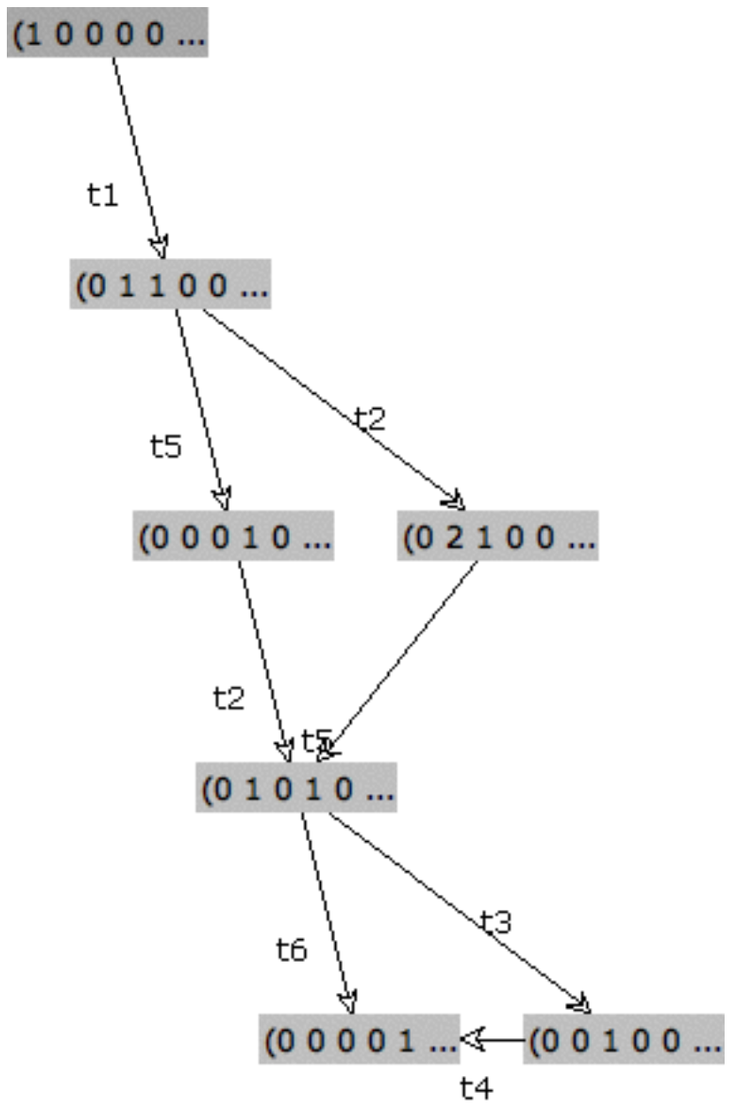
N can be sound even if N* is not S-coverable

# Example: N sound but N* not S-coverable



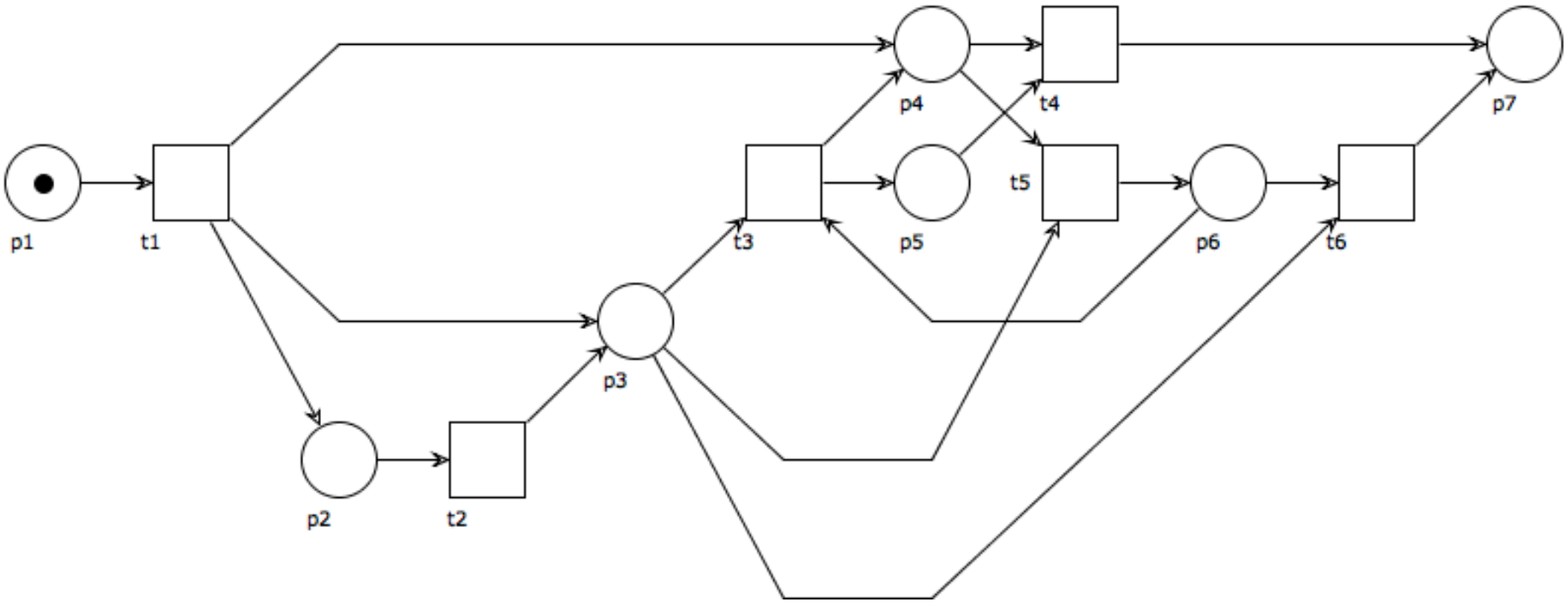
# Example: N sound but N* not S-coverable



# Exercises

Find all (maximal) S-components using WoPeD



# Exercises

Draw a workflow net N that is S-coverable but such that N* is not sound

# S-Coverability diagnosis

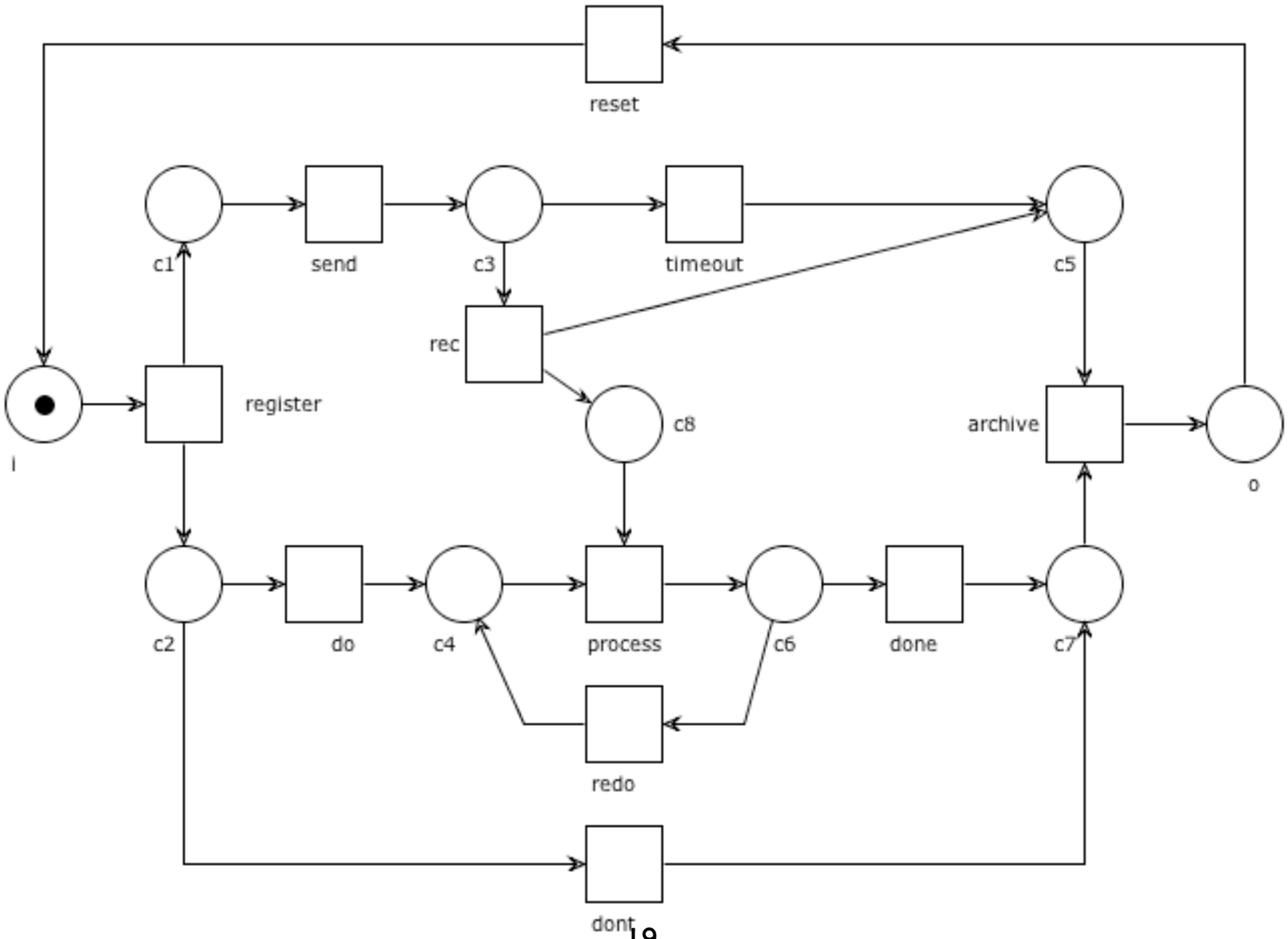
If N* is free-choice, live and bounded
it must be S-coverable (S-coverability theorem)

(note that any S-component of N* includes i, o, reset, by strong-connectedness)

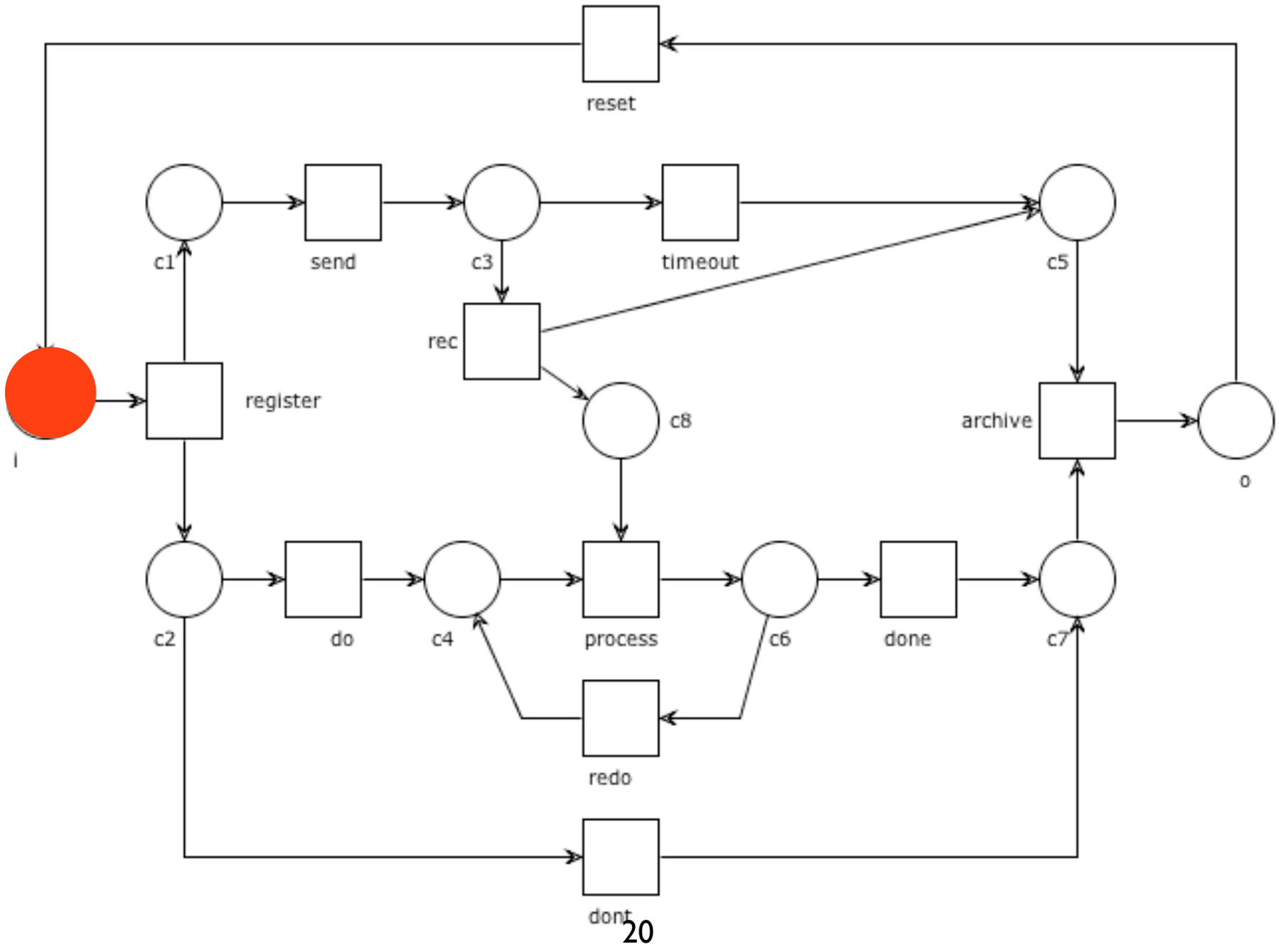
**Corollary:** If N is sound and free-choice,
then N* must be S-coverable

**N free-choice + N* not S-coverable => N not sound**

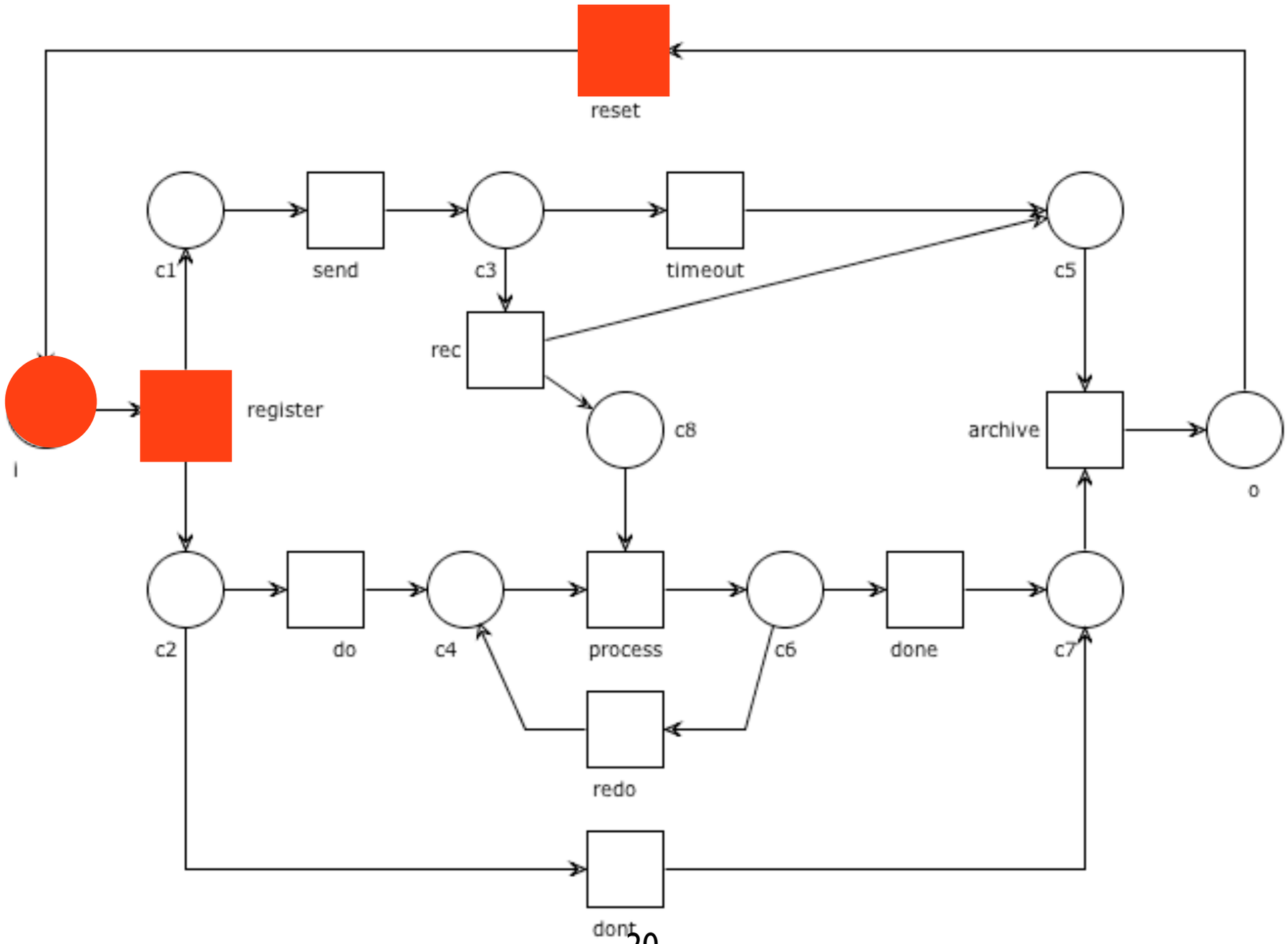
# Running example: S-cover for N* ?



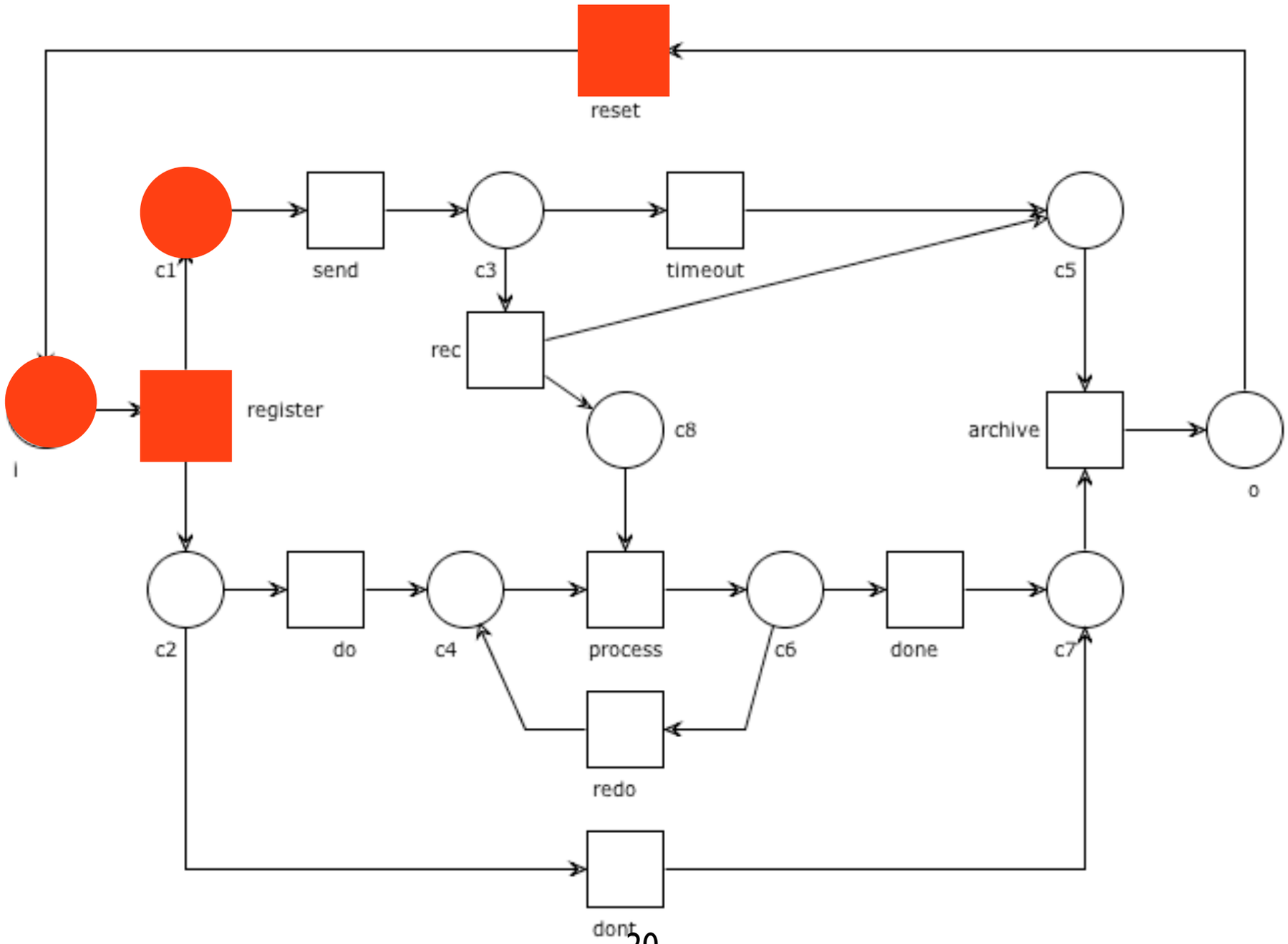
# Running example: S-cover for N* ?



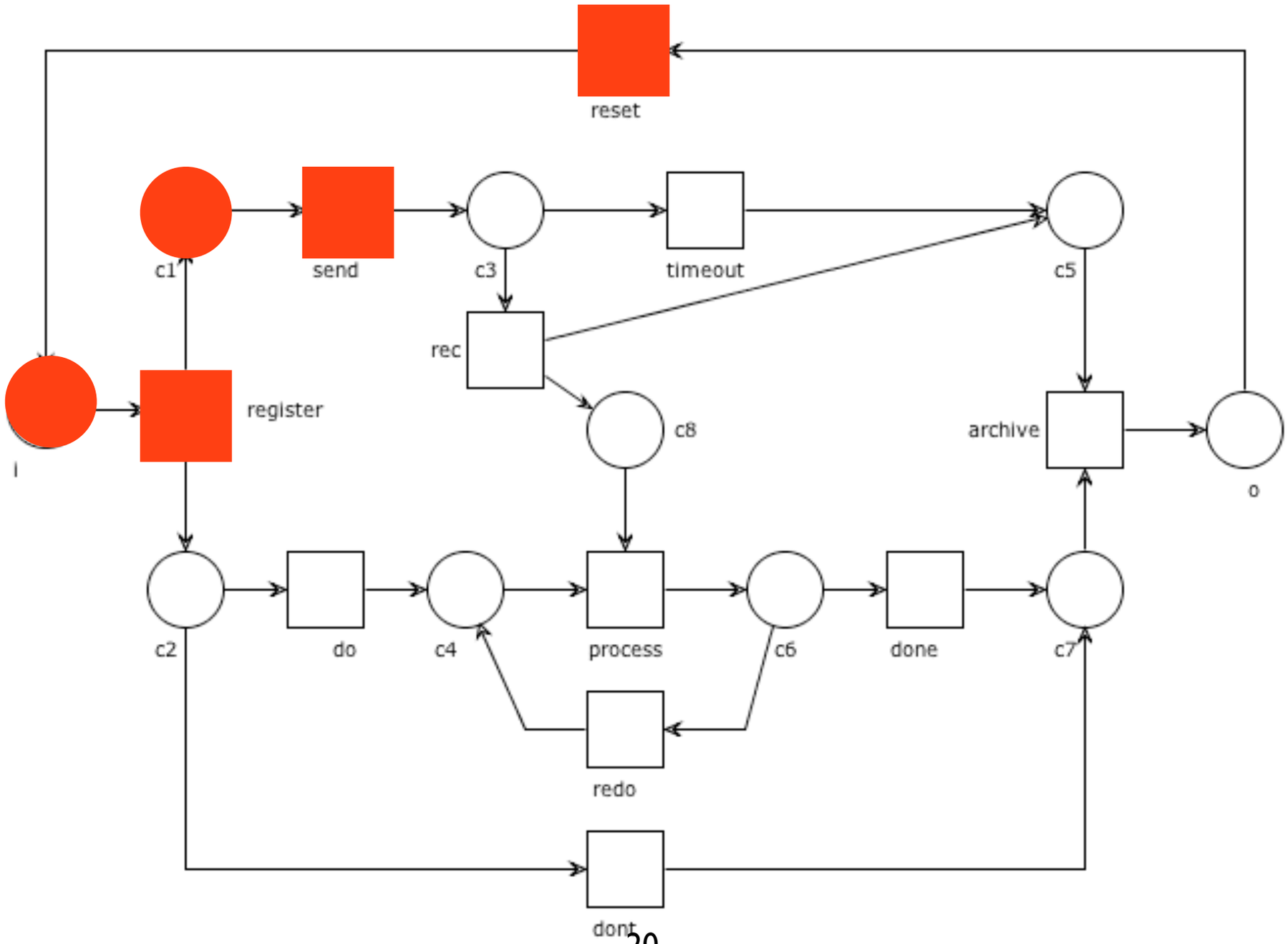
# Running example: S-cover for N* ?



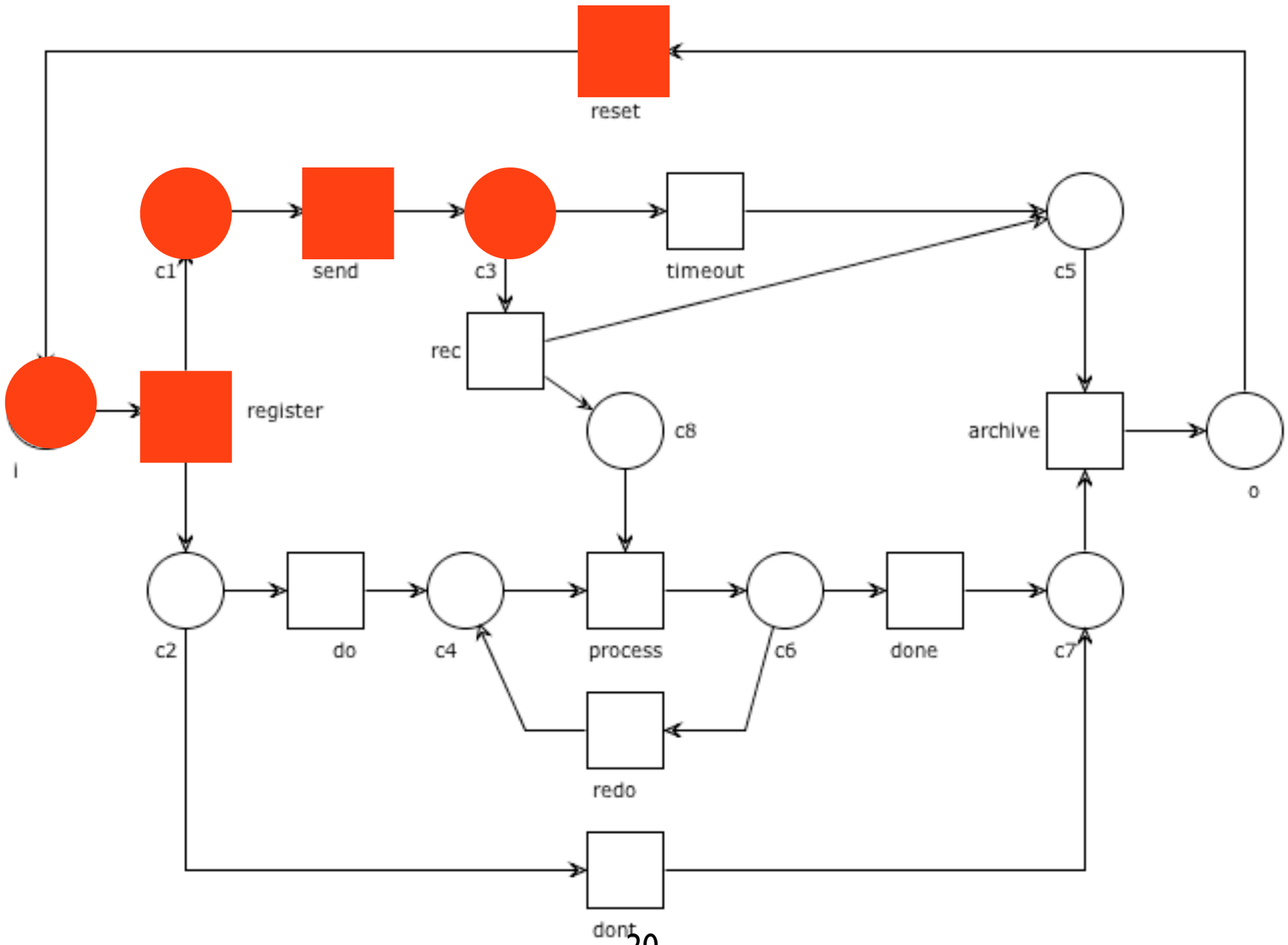
# Running example: S-cover for N* ?



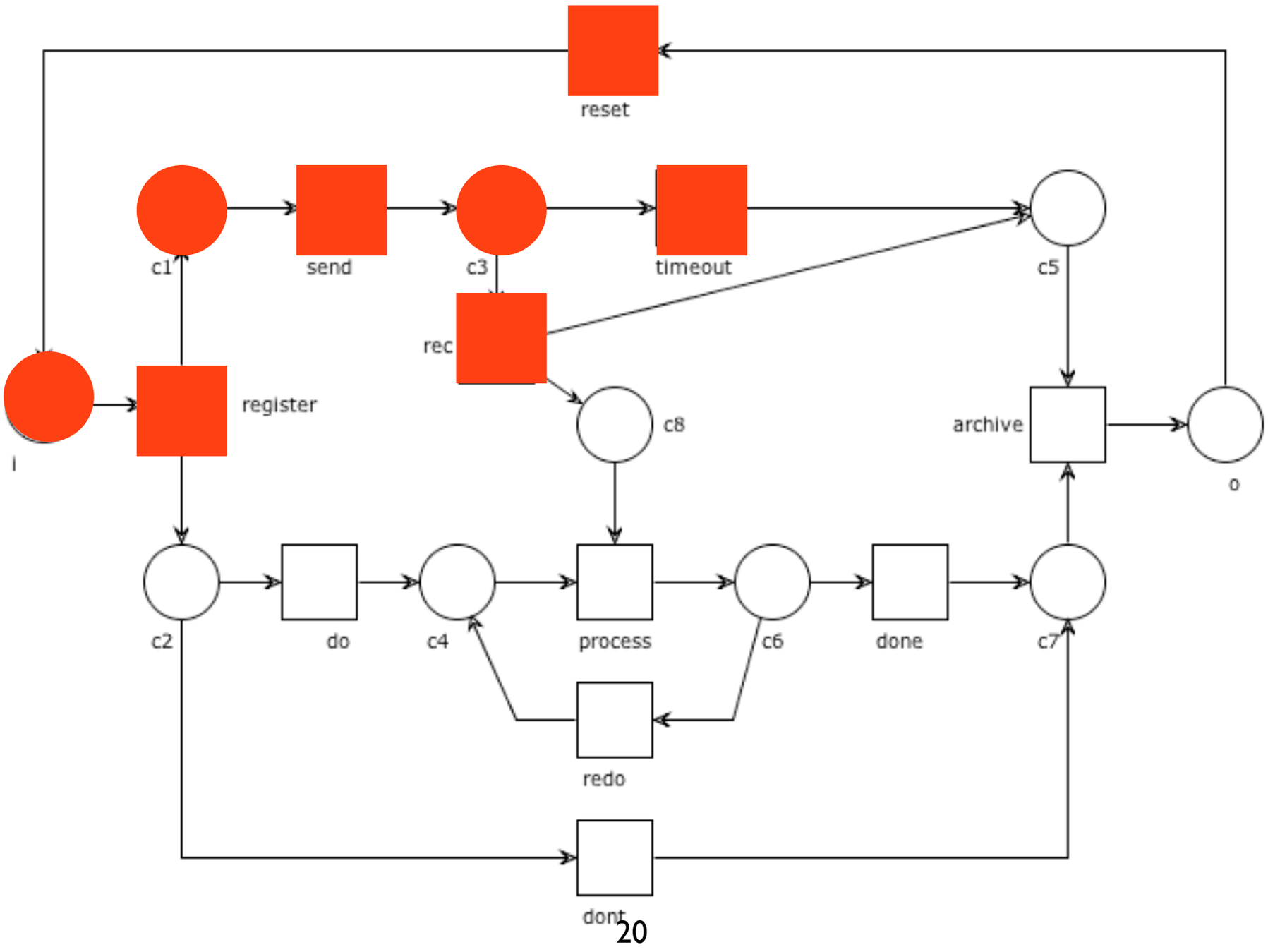
# Running example: S-cover for N* ?



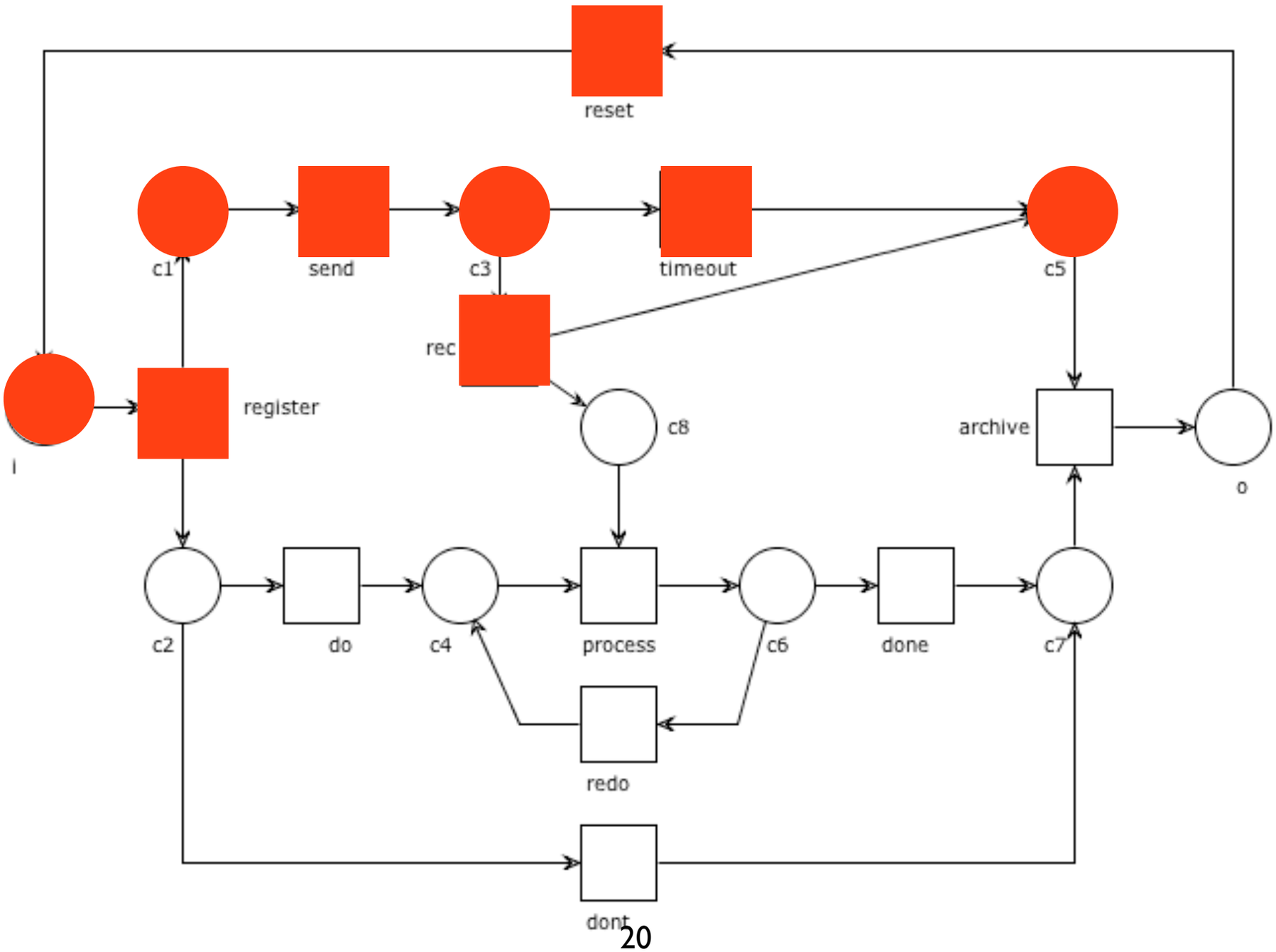
# Running example: S-cover for N* ?



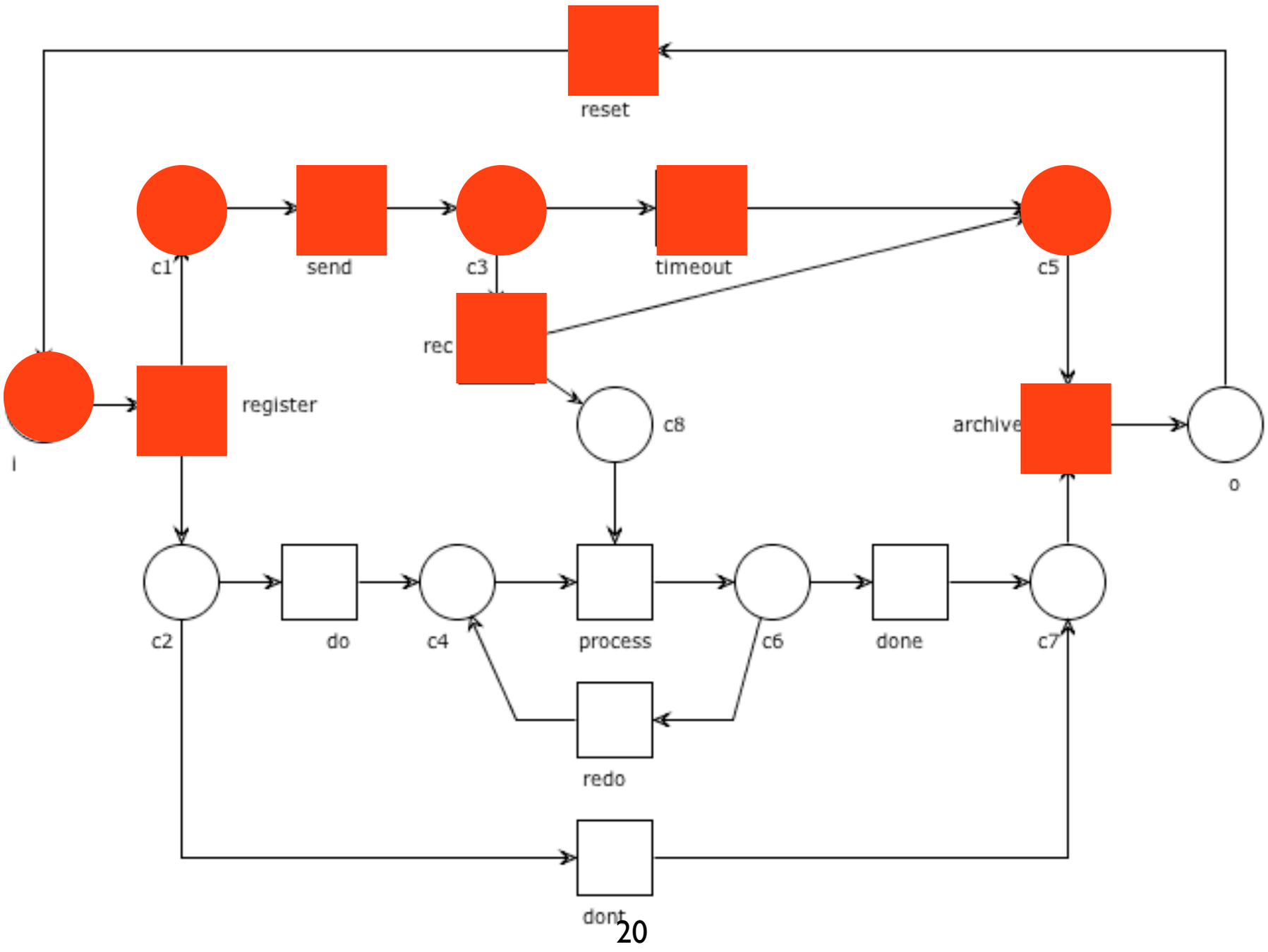
# Running example: S-cover for N* ?



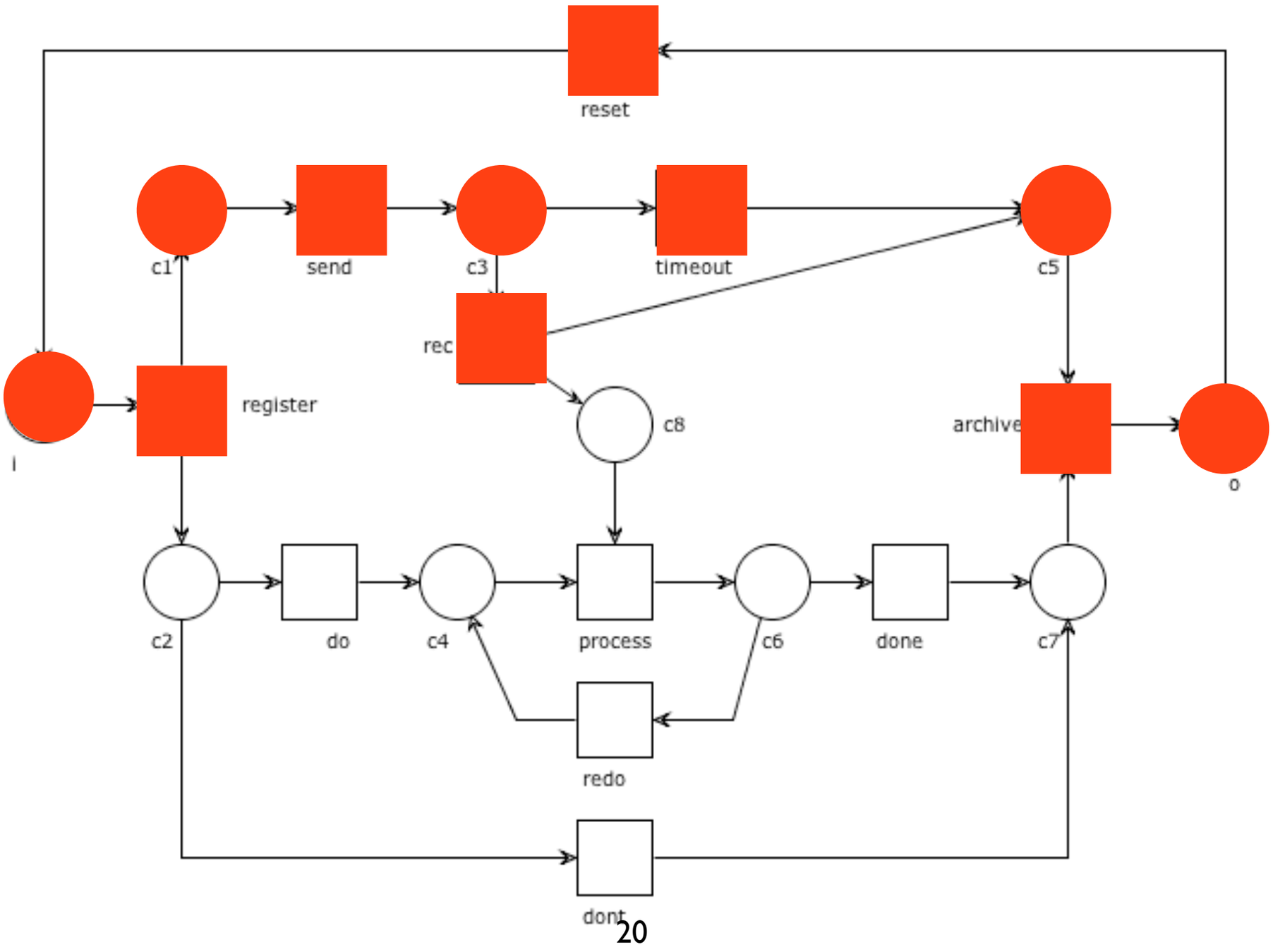
# Running example: S-cover for N* ?



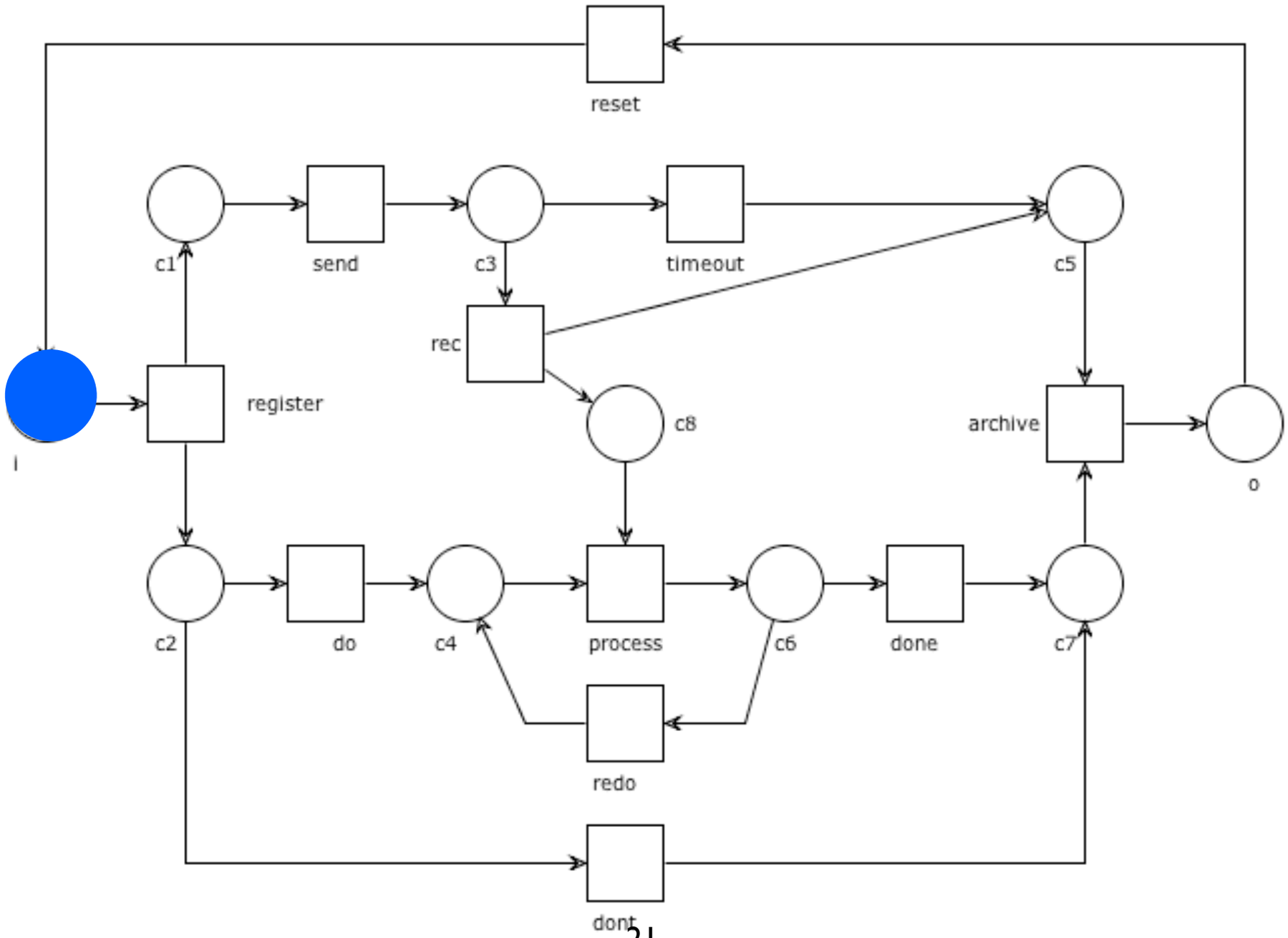
# Running example: S-cover for N* ?



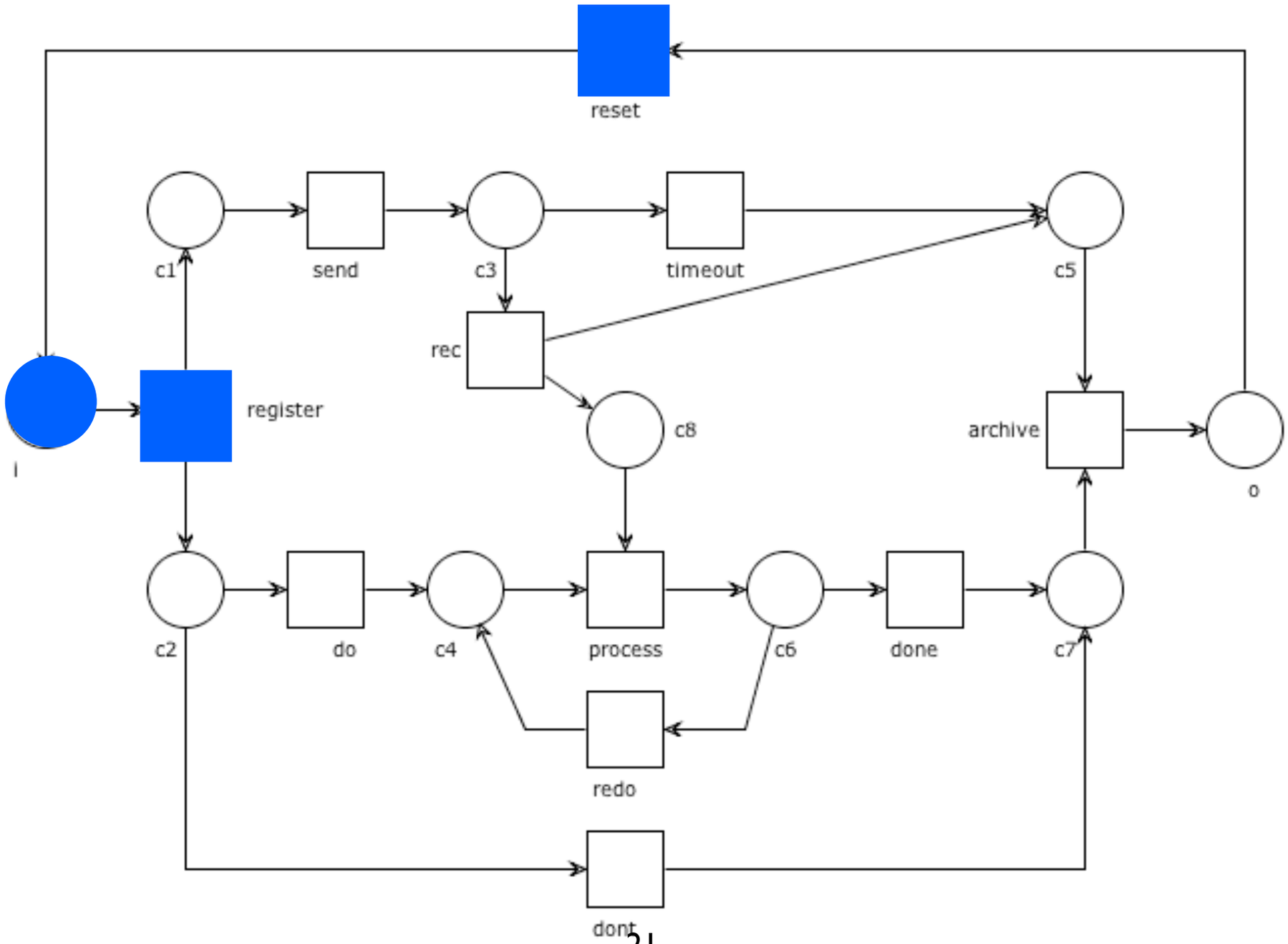
# Running example: S-cover for N* ?



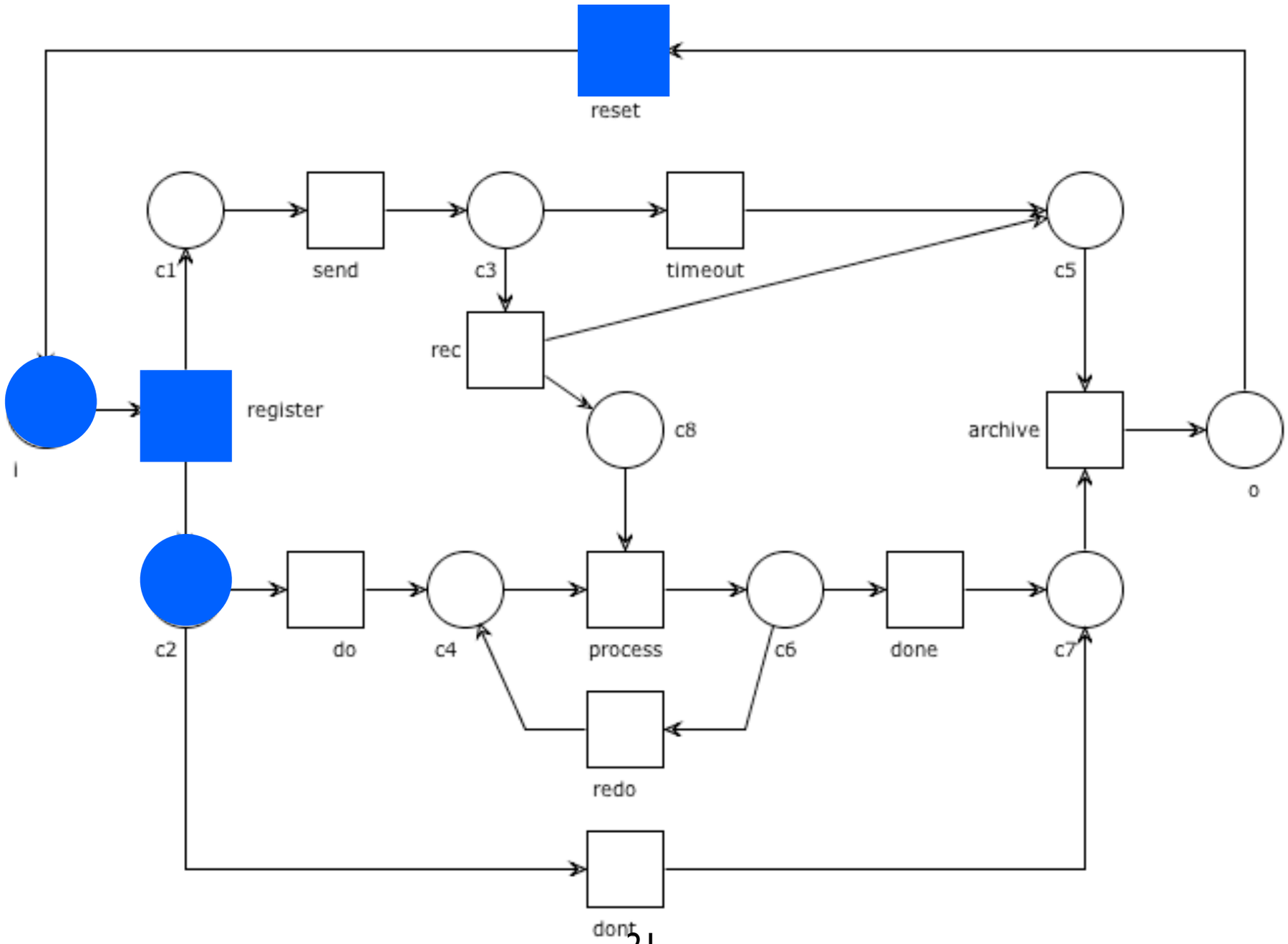
# Running example: S-cover for N*



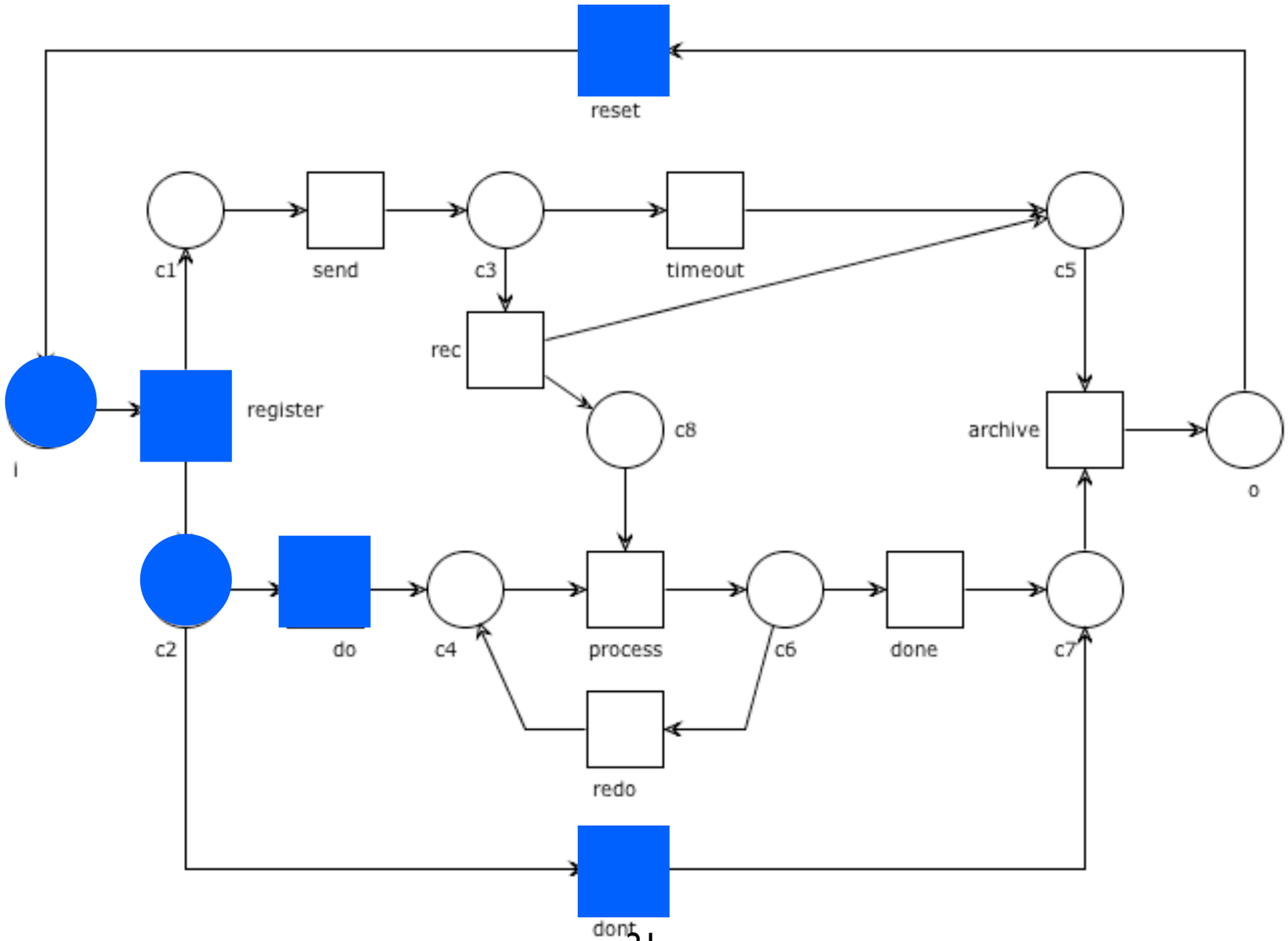
# Running example: S-cover for N*



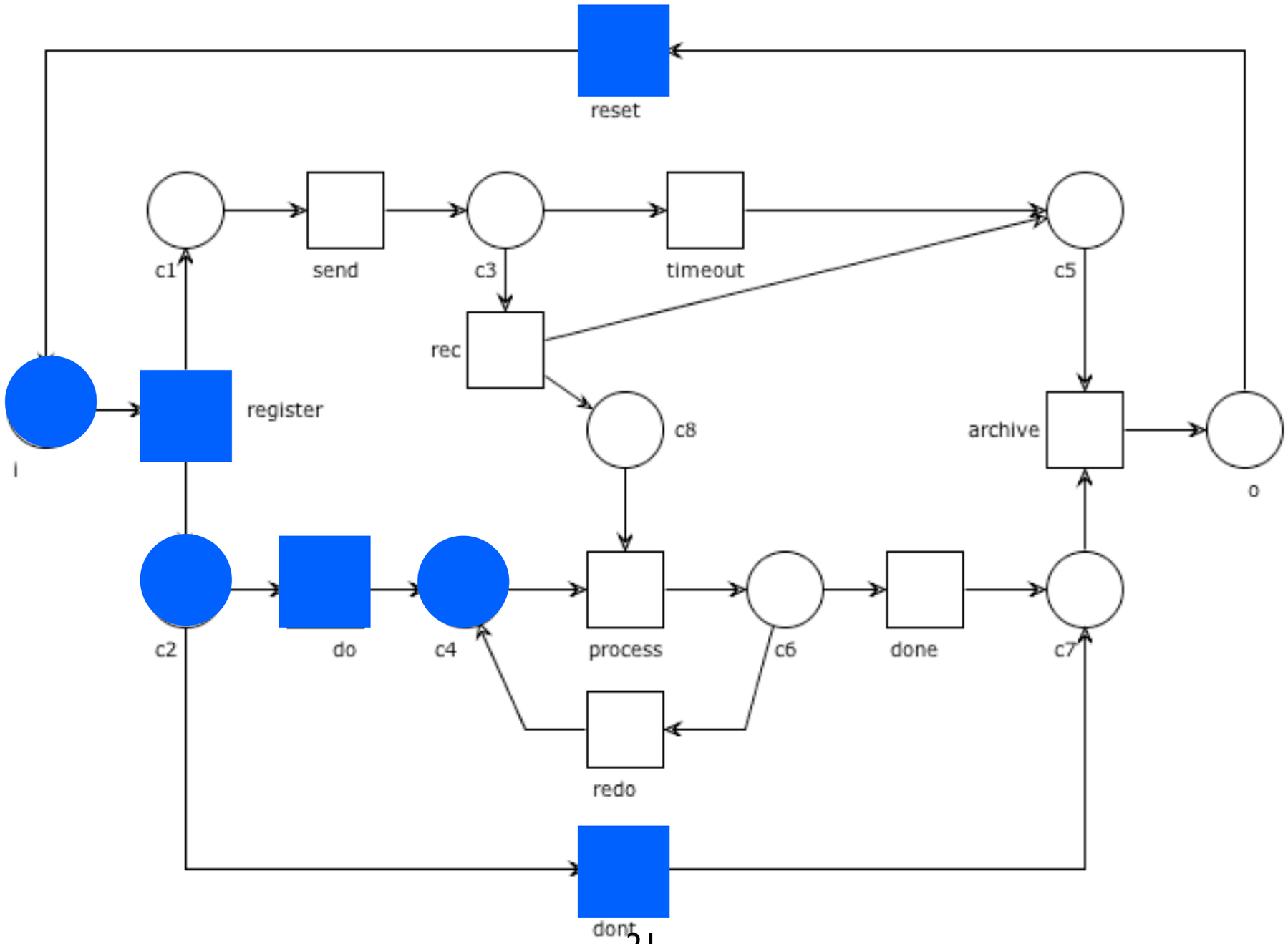
# Running example: S-cover for N*



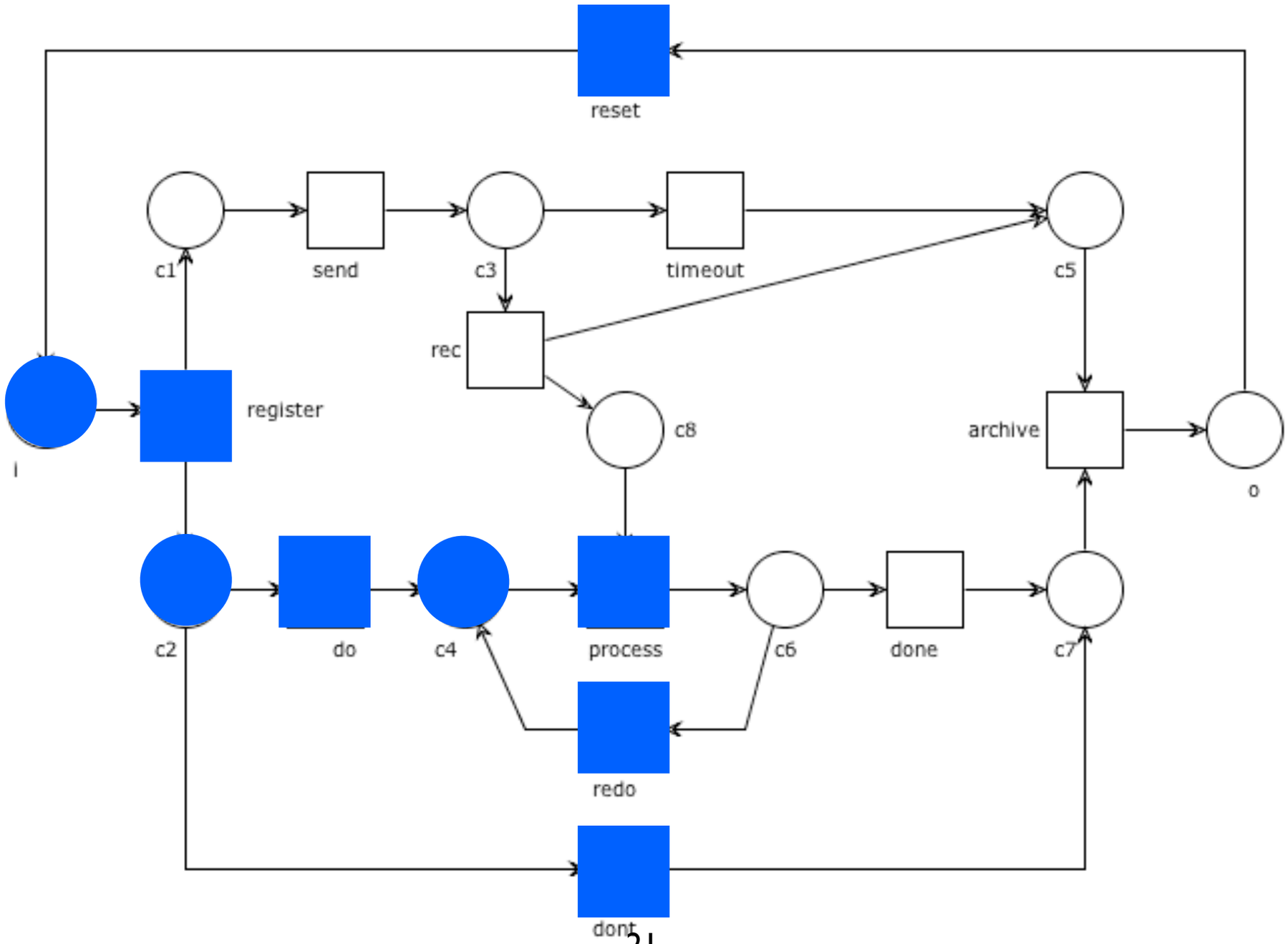
# Running example: S-cover for N*



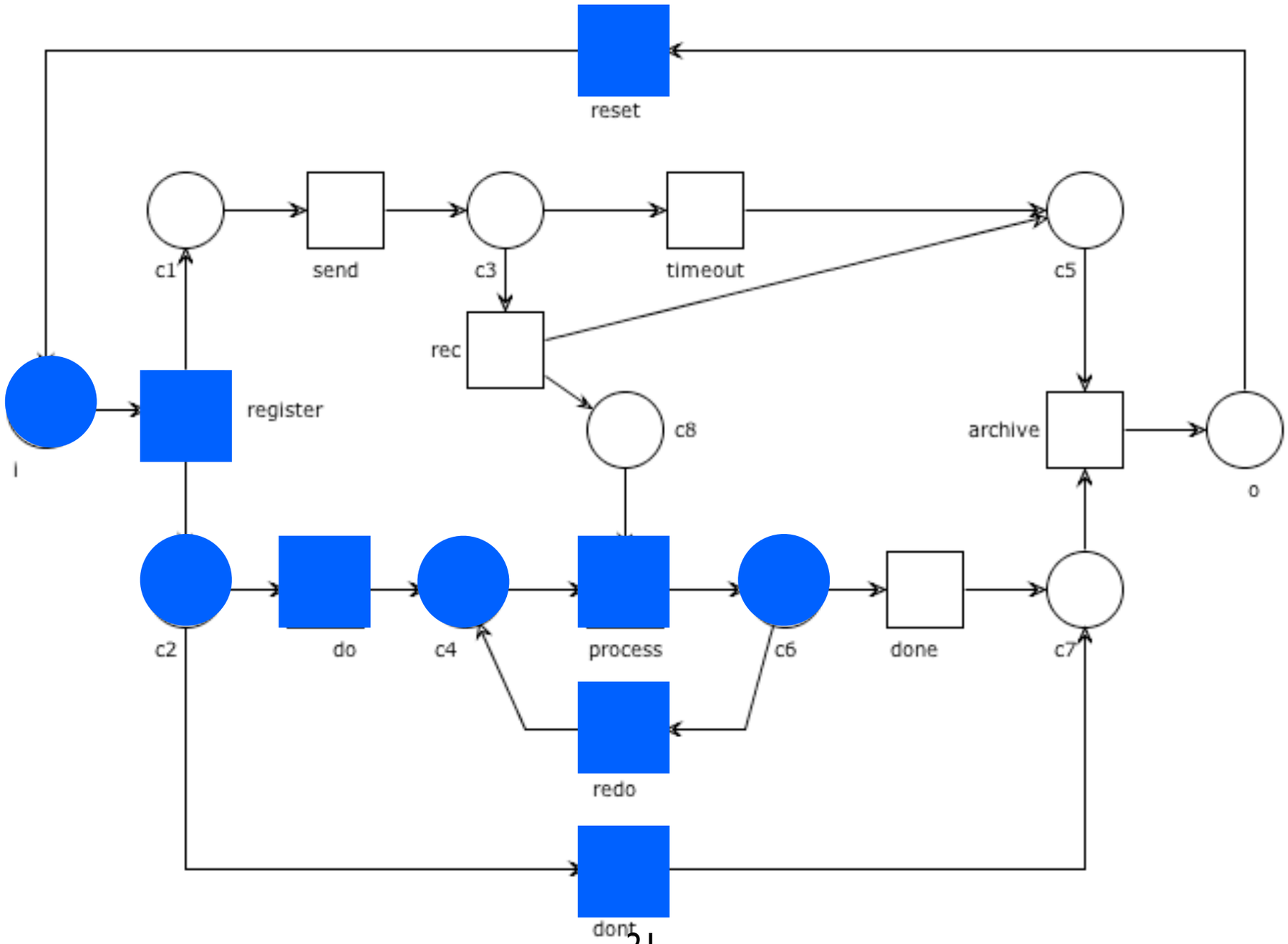
# Running example: S-cover for N*



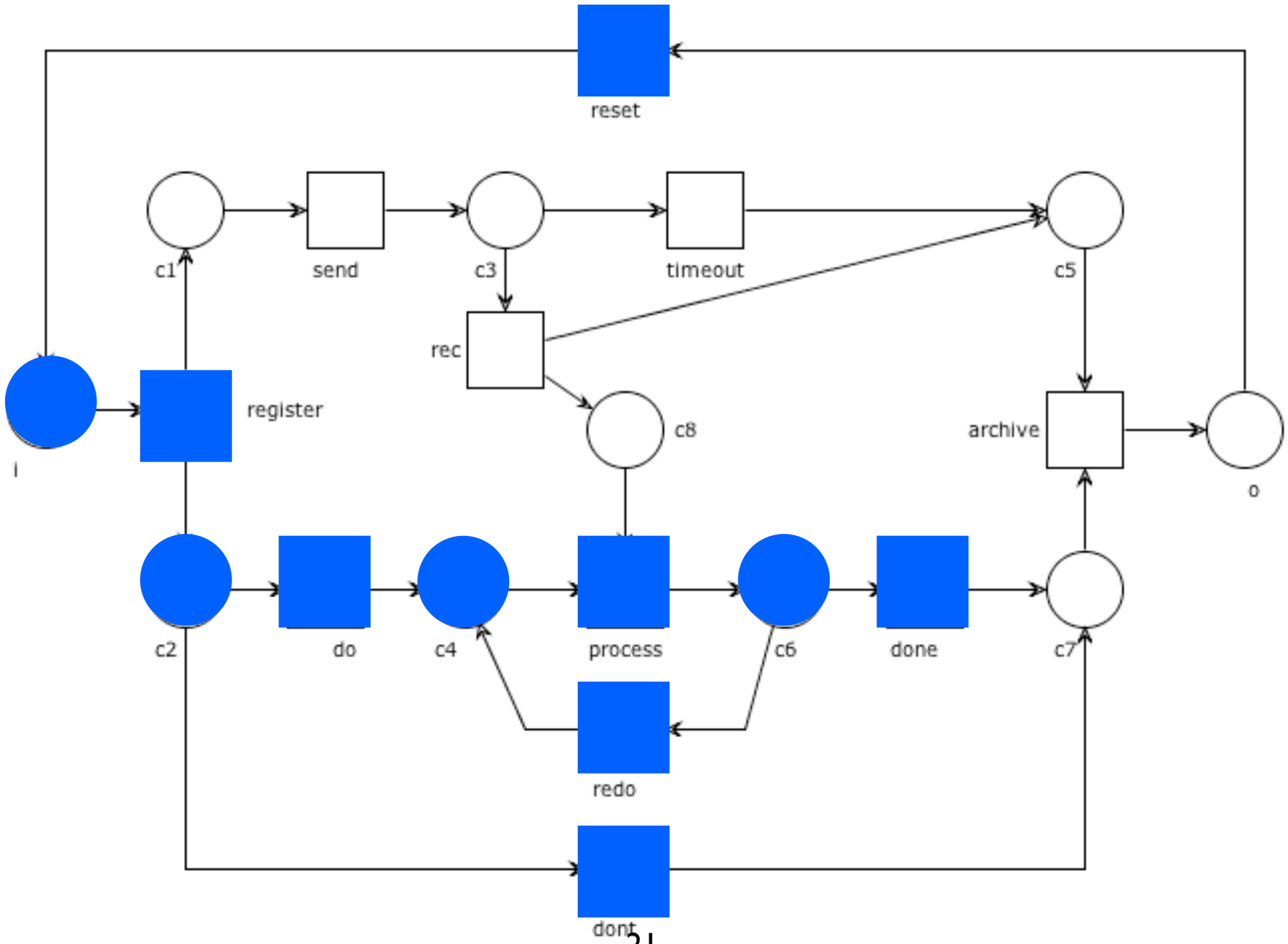
# Running example: S-cover for N*



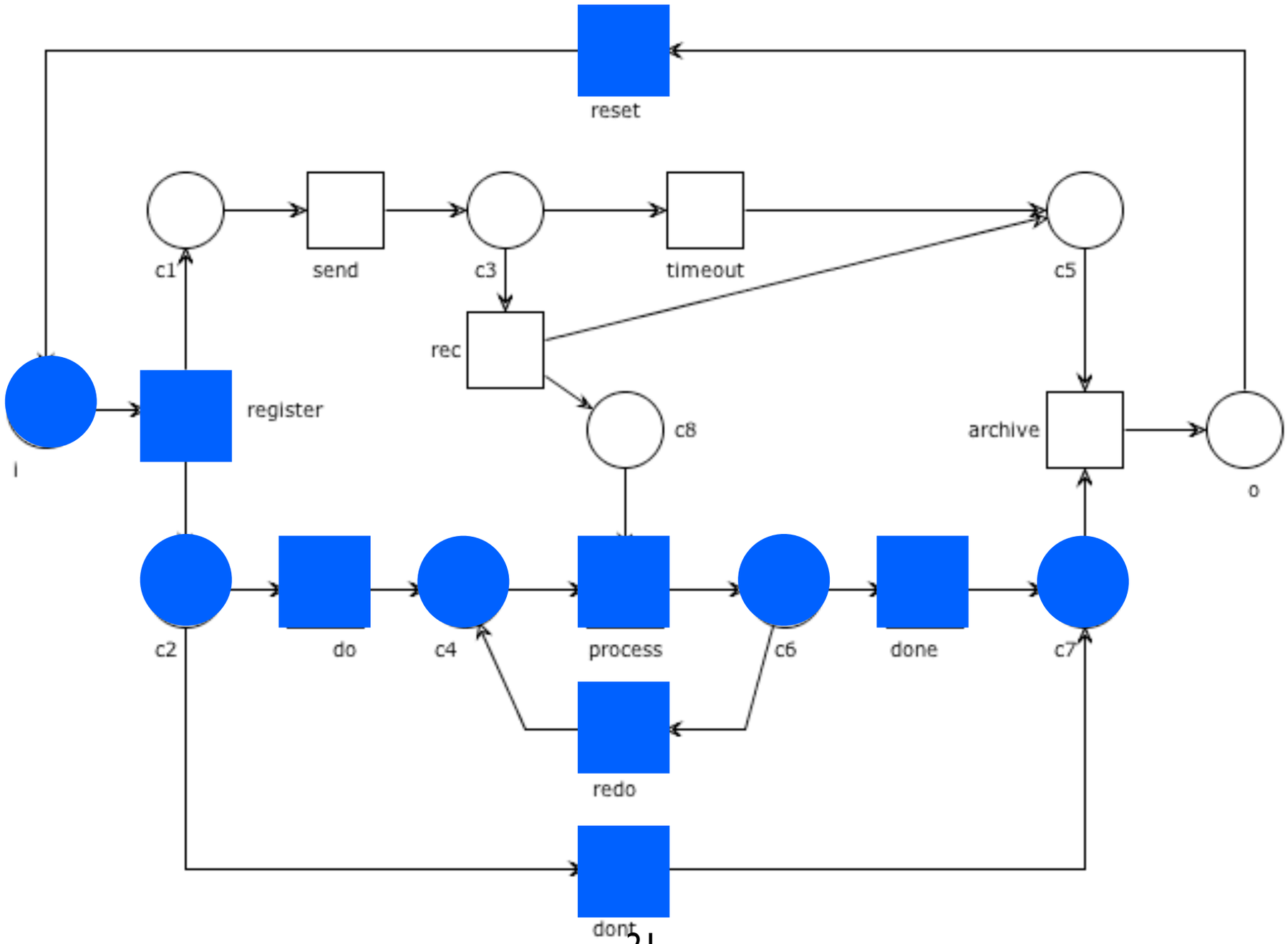
# Running example: S-cover for N*



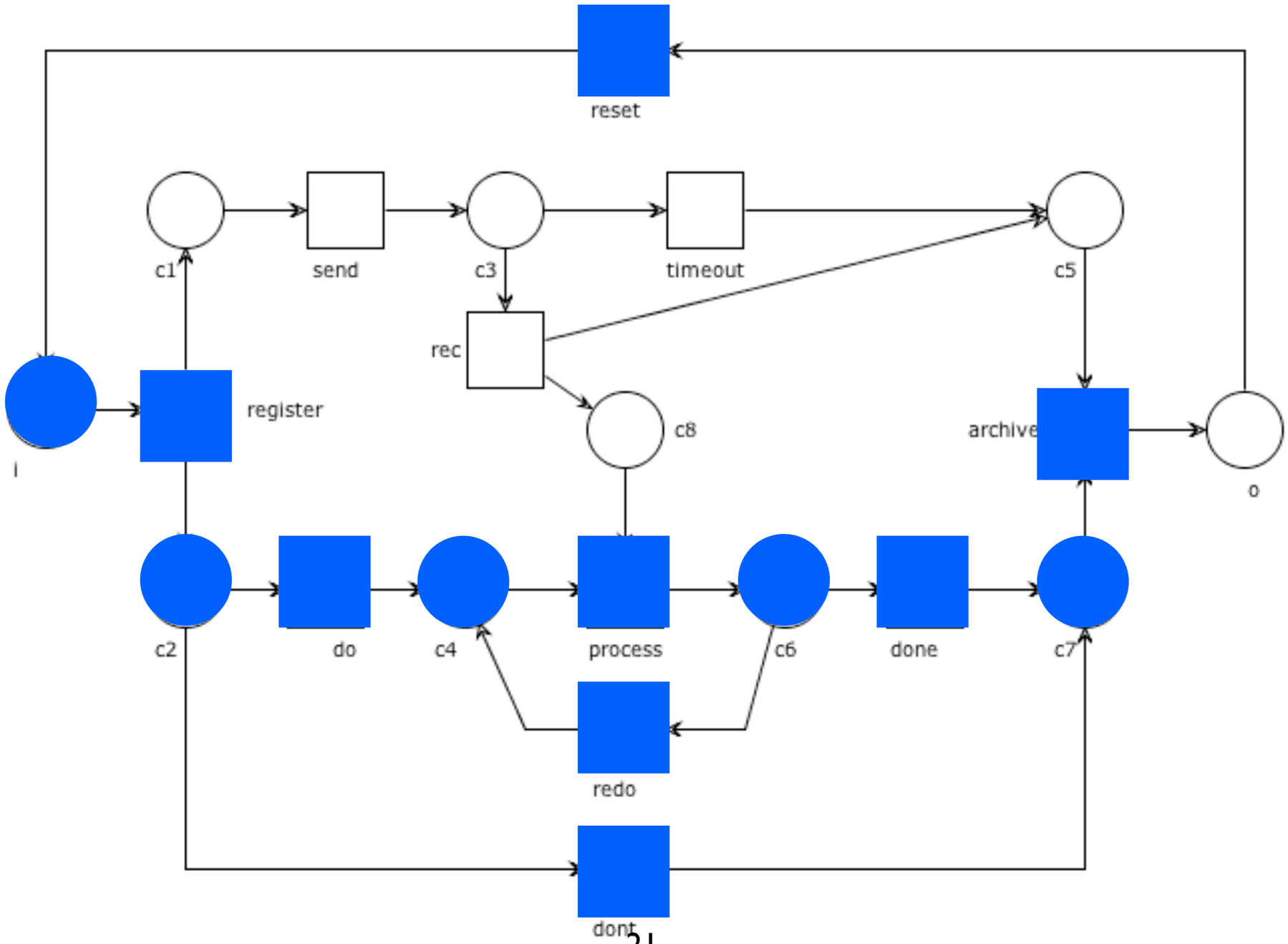
# Running example: S-cover for N*



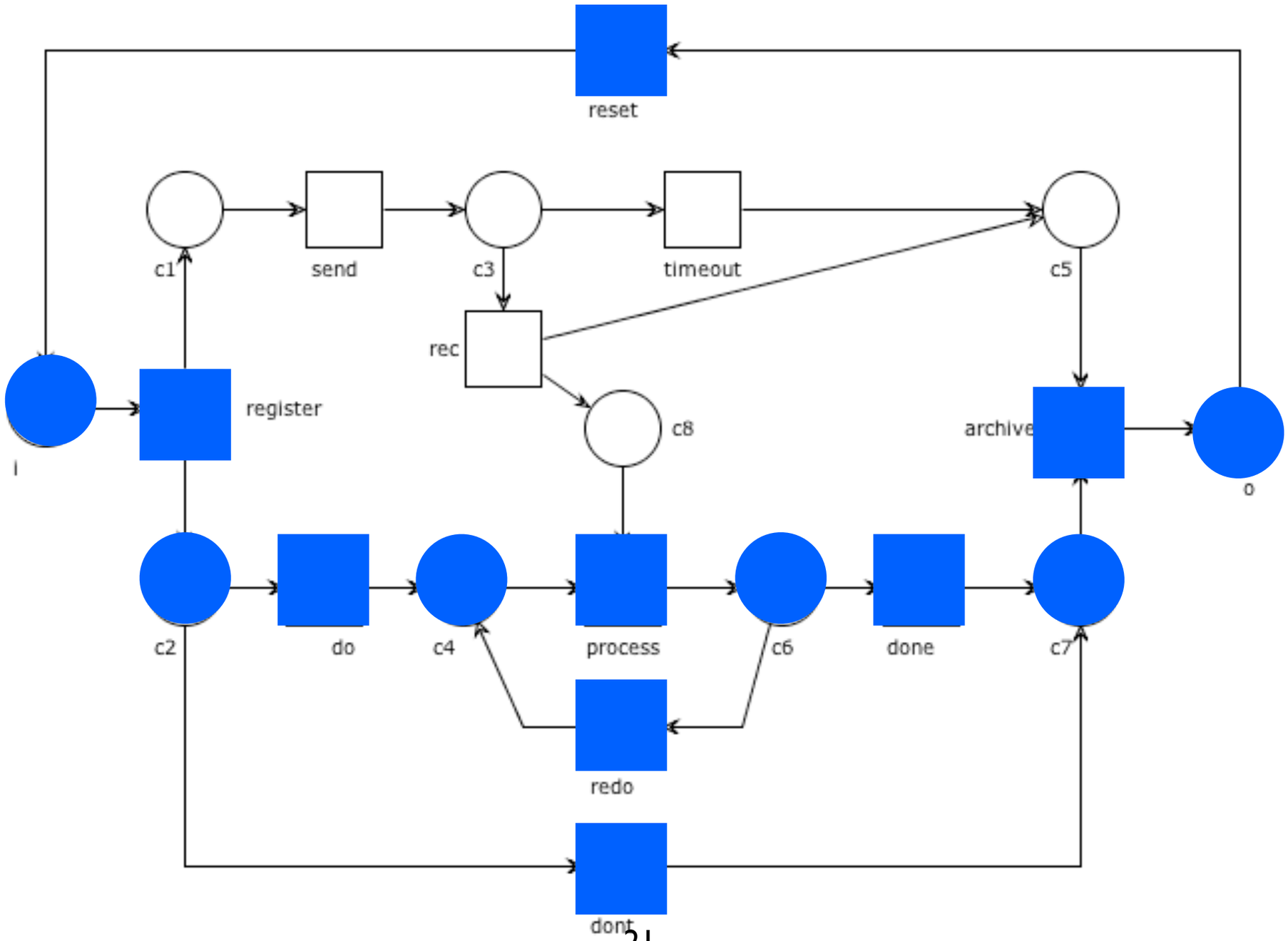
# Running example: S-cover for N*



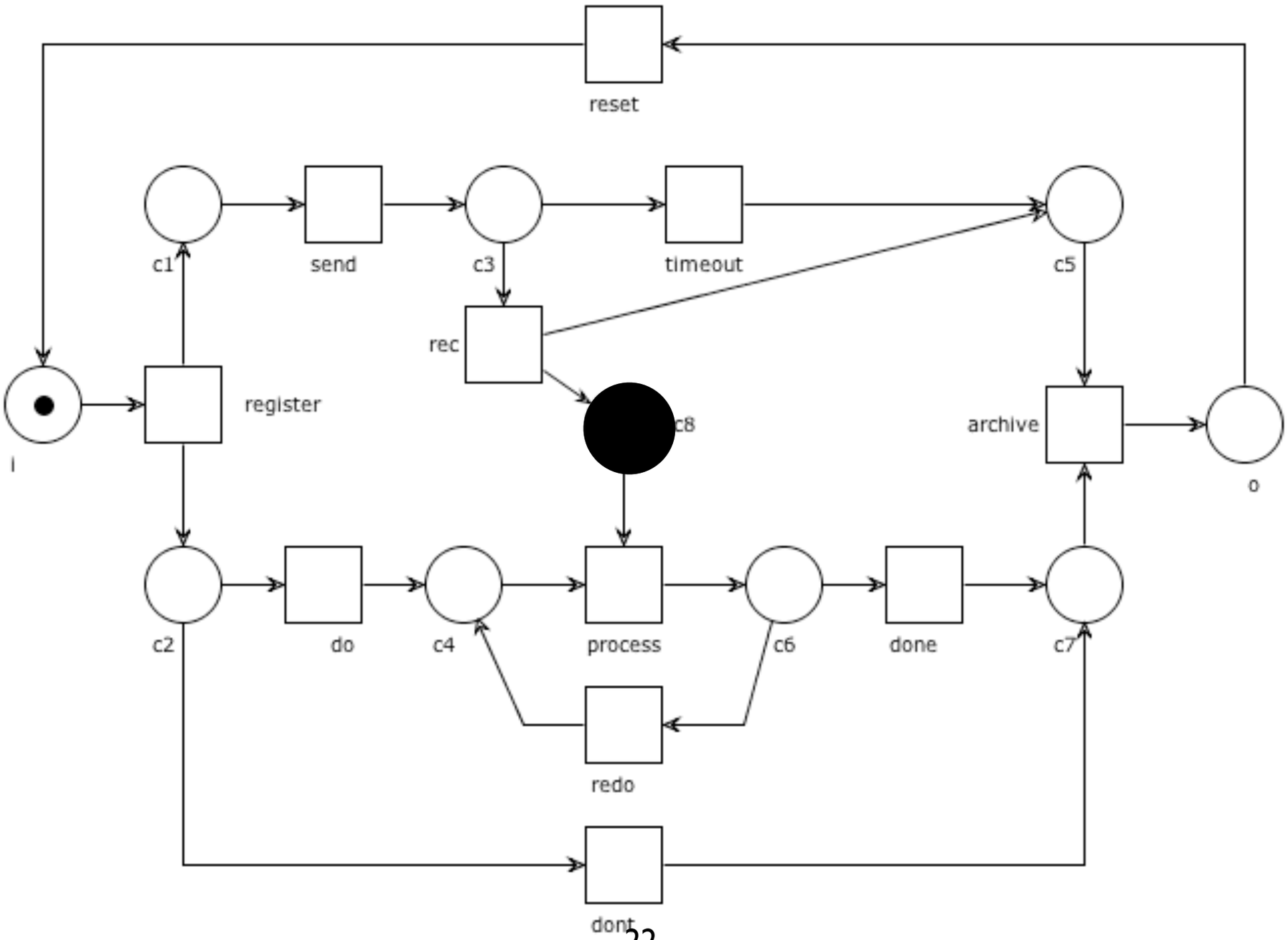
# Running example: S-cover for N*



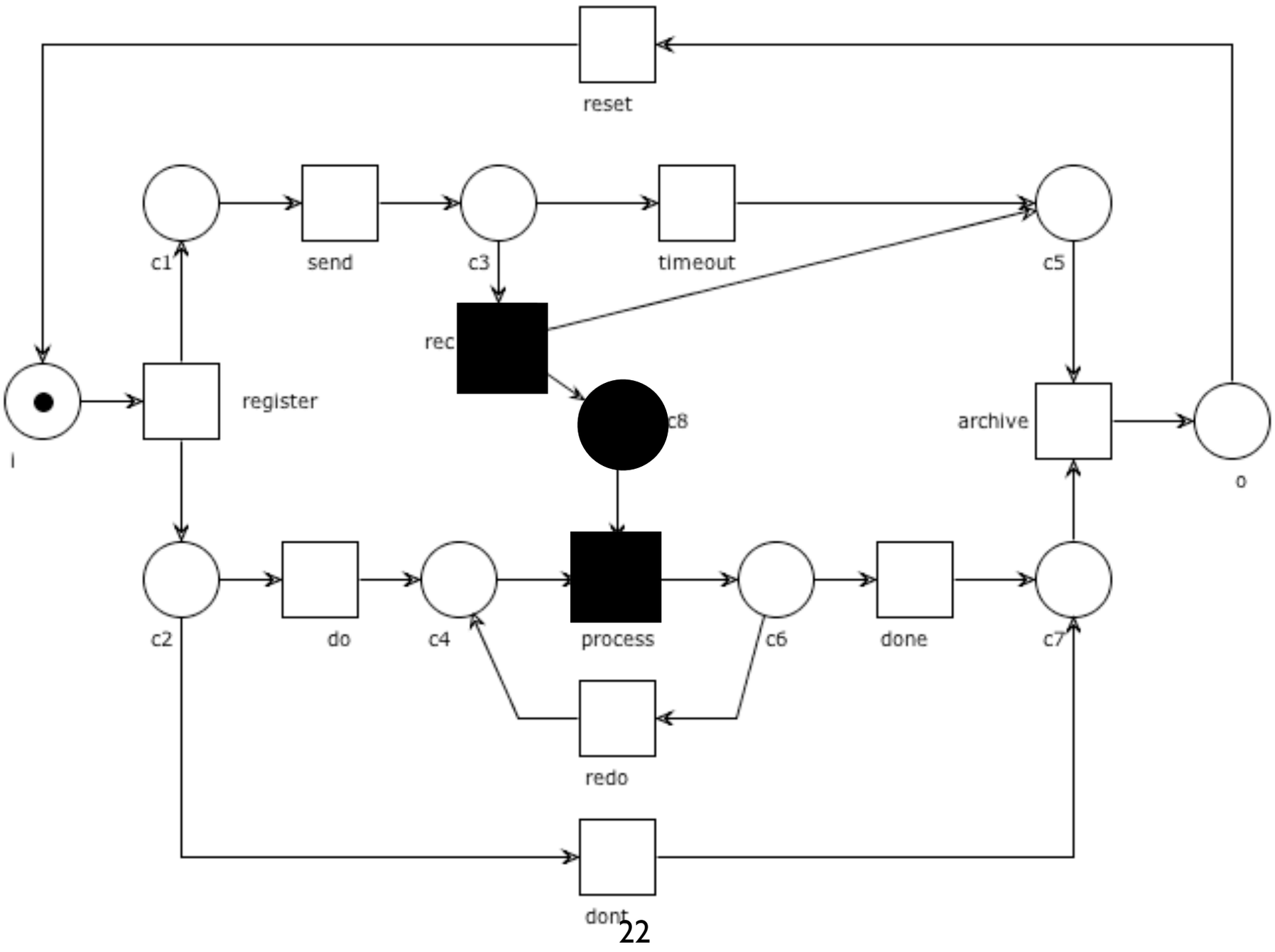
# Running example: S-cover for N*



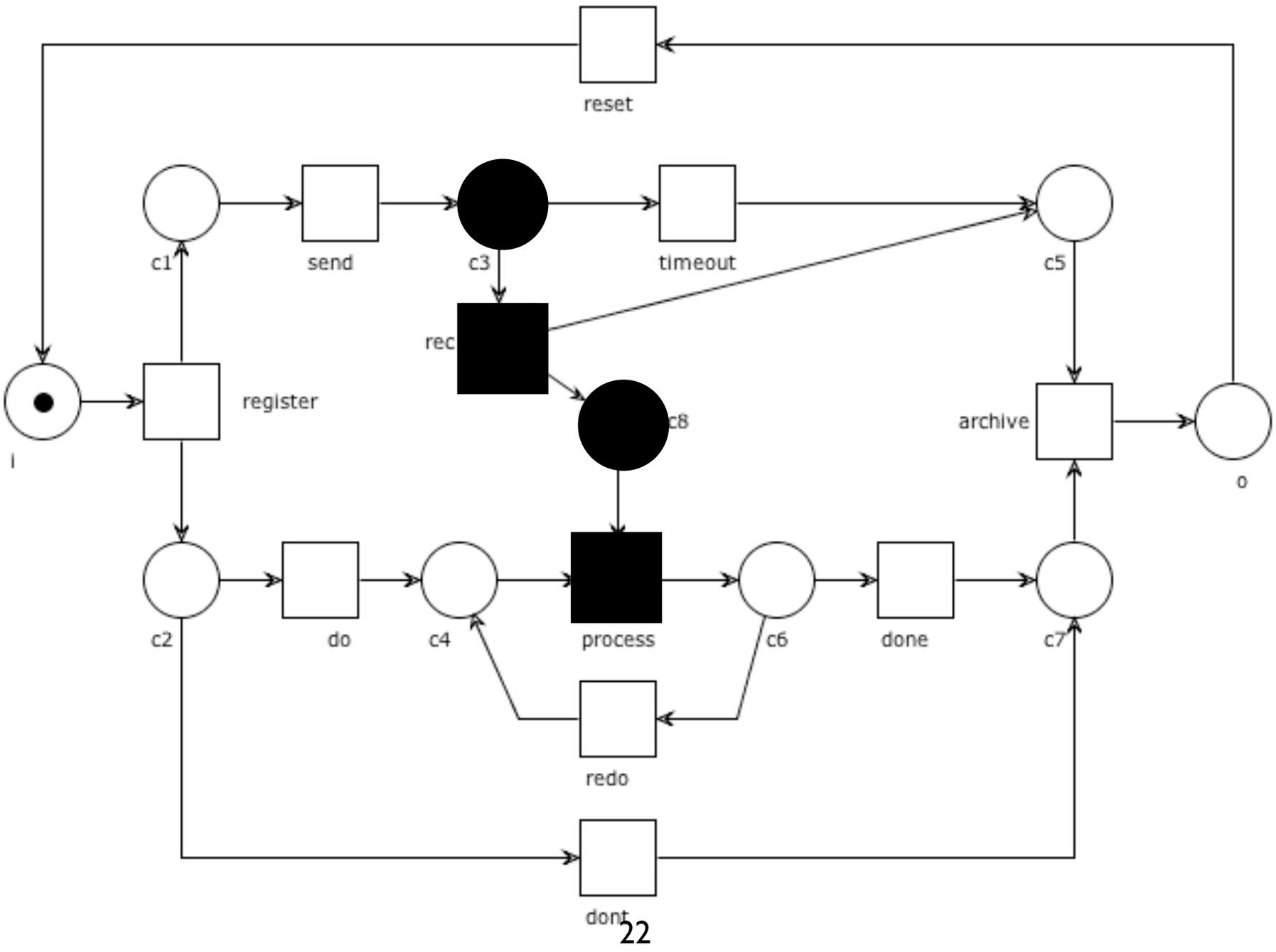
# Running example: S-cover for N*



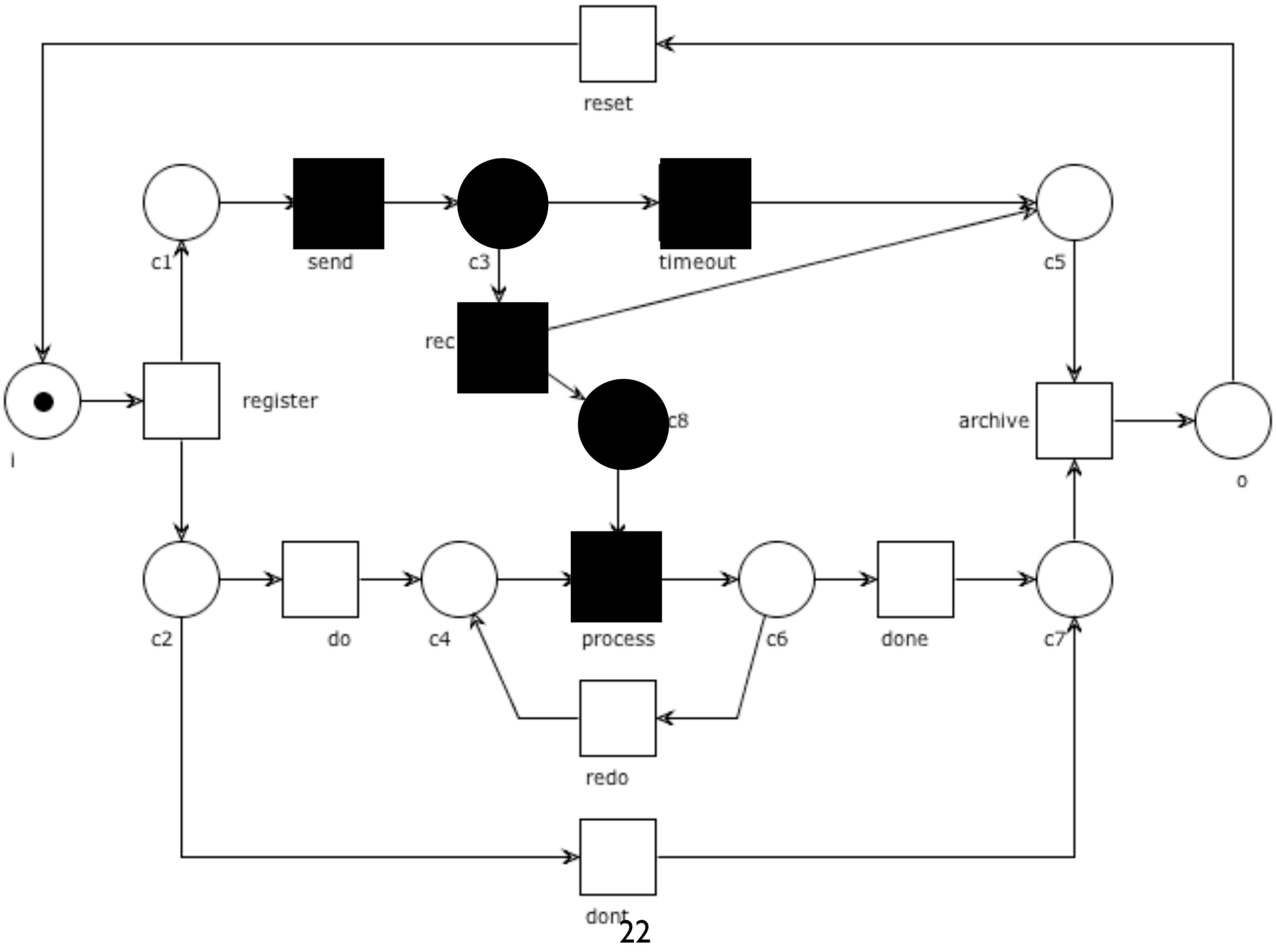
# Running example: S-cover for N*



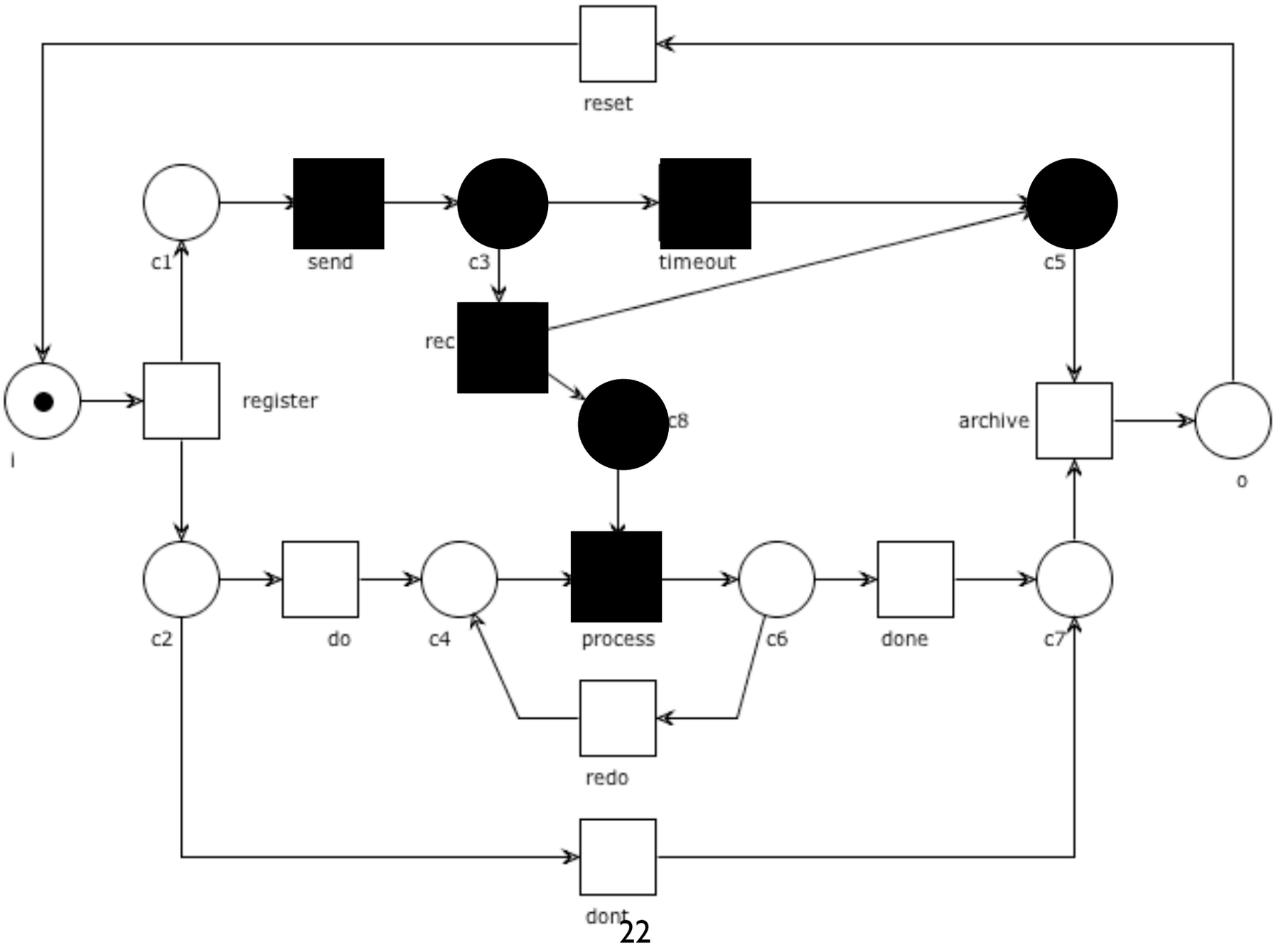
# Running example: S-cover for N*



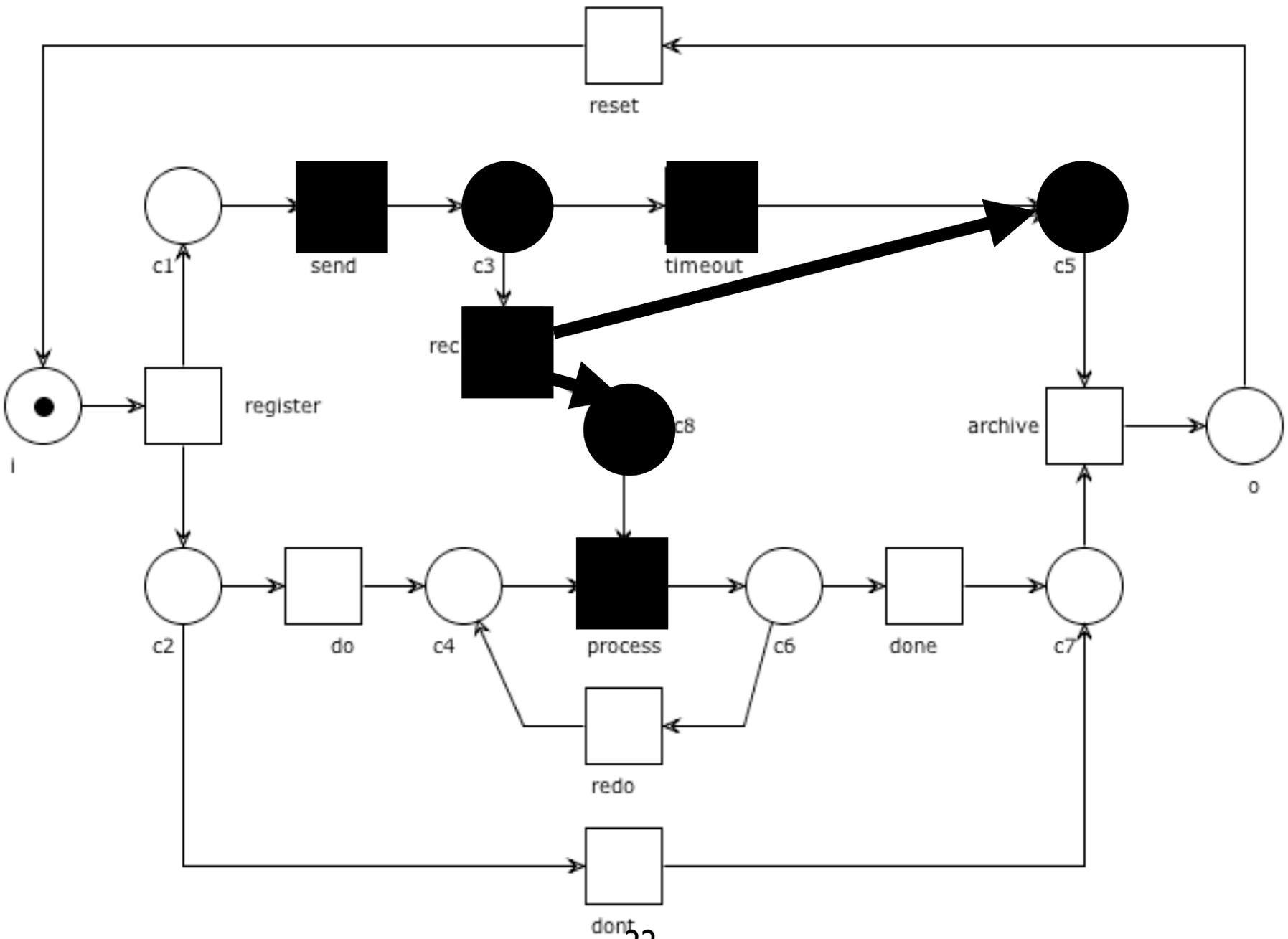
# Running example: S-cover for N*



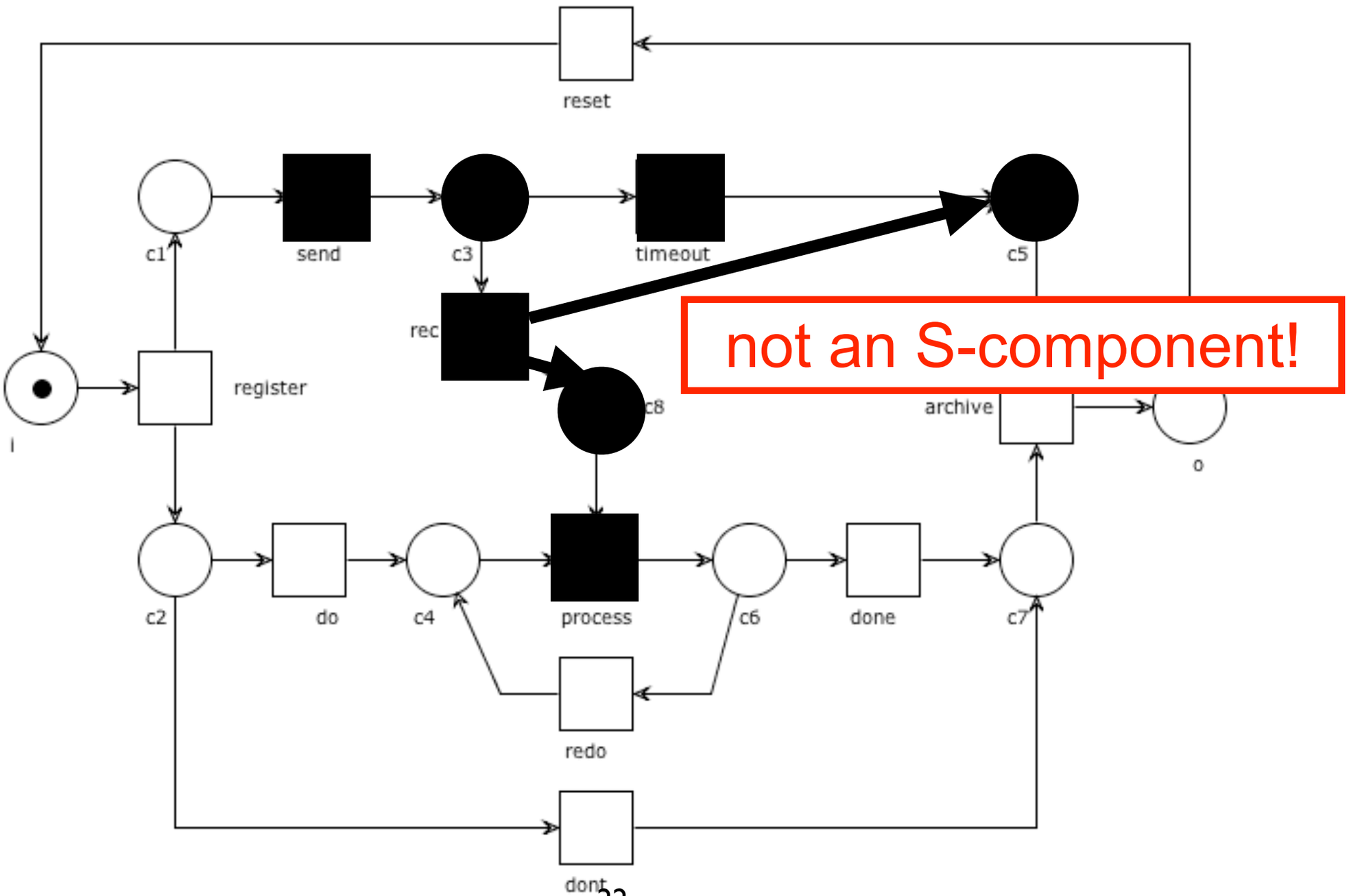
# Running example: S-cover for N*



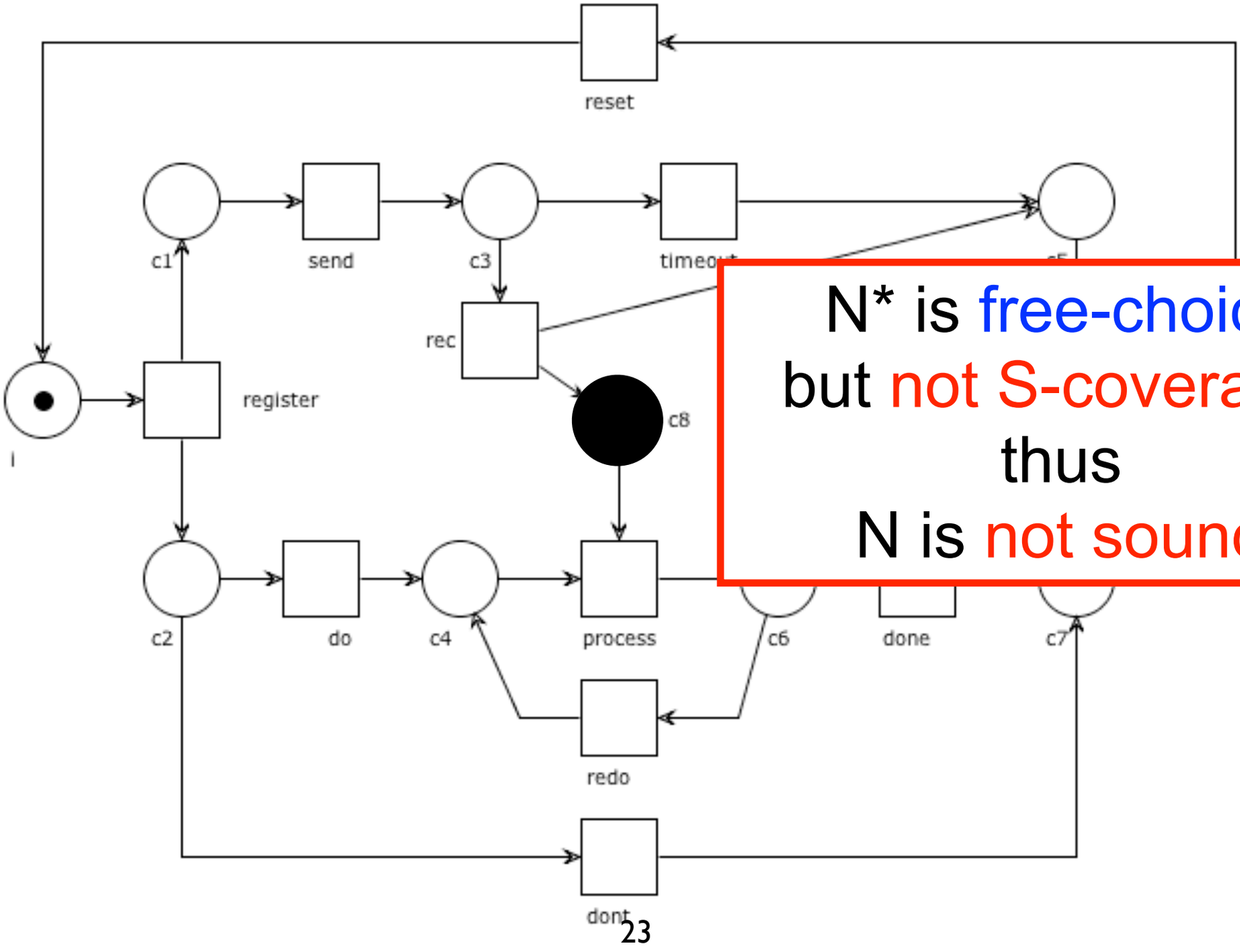
# Running example: S-cover for N*



# Running example: S-cover for N*

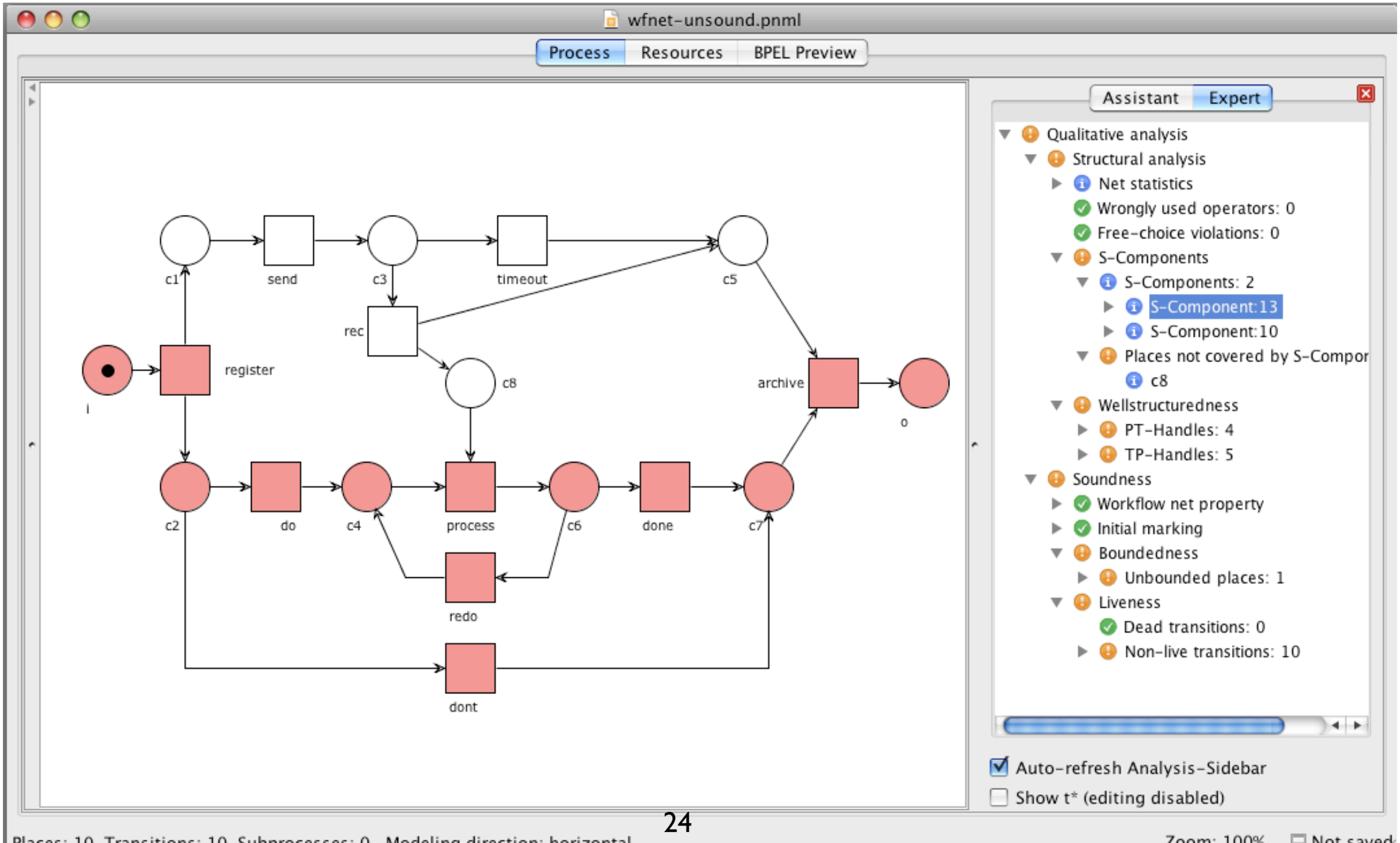


# Running example: S-cover for N* ? No




N* is free-choice
but not S-coverable
thus
N is not sound

# Running Example: WoPeD Diagnosis



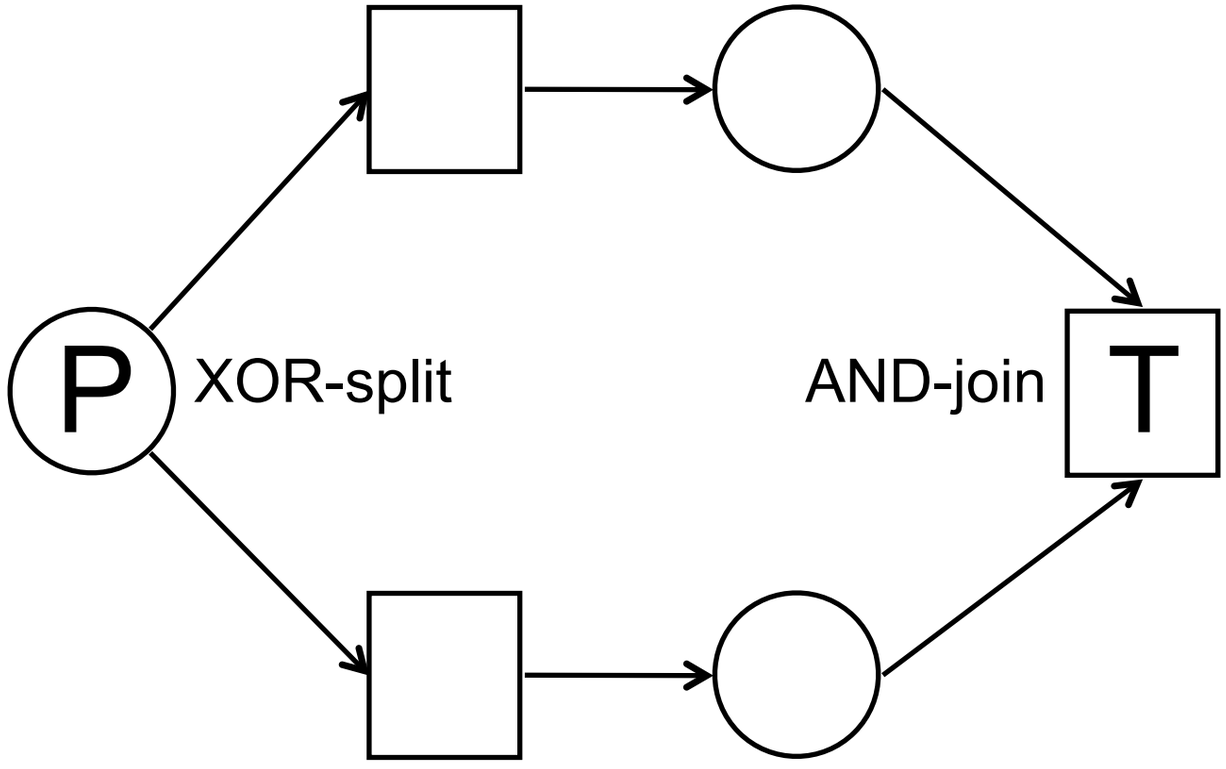
# Split / Join Balancing

A good workflow design is characterized by a balance between AND/XOR-split and AND/XOR-joins

**Any mismatch is a potential source of errors**

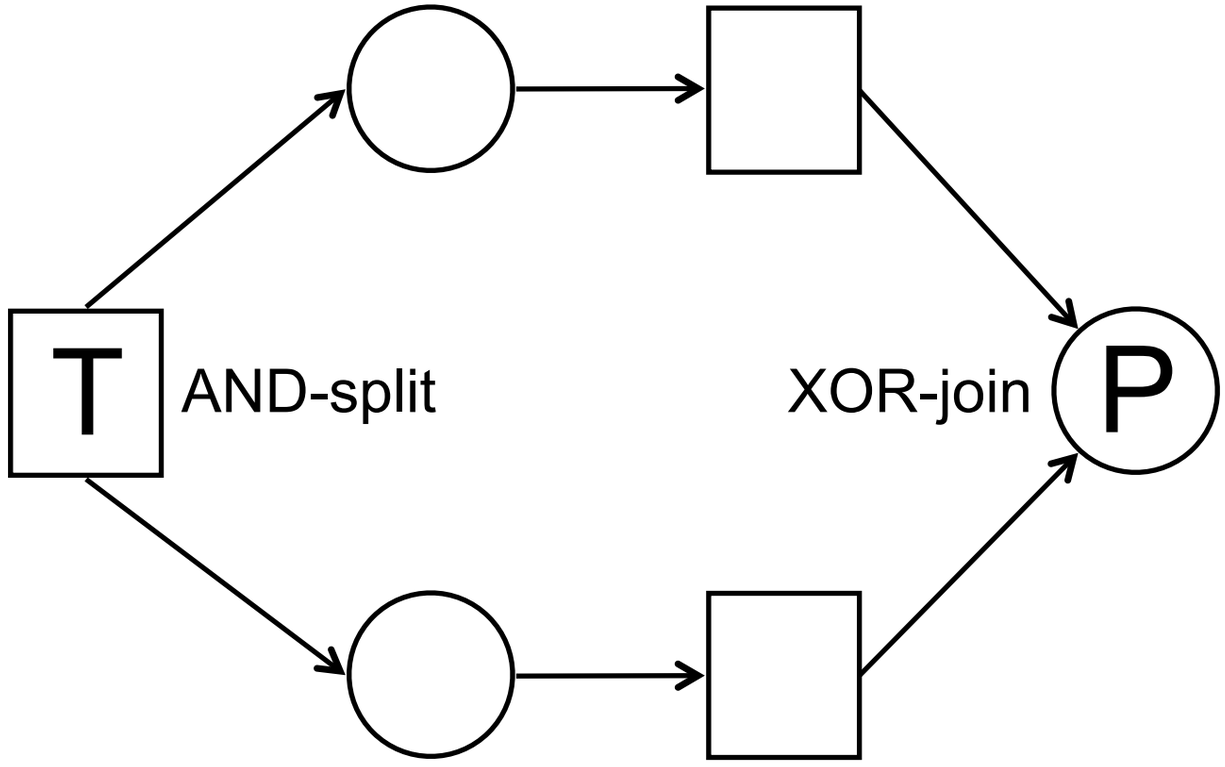
# PT-handles

Two alternative flows created via a XOR-split should not be synchronized by an AND-join **(the net could deadlock)**



# TP-handles

Two parallel flows initiated by an AND-split
should not be joined by a XOR-join
**(multiple tokens can be produced in the same place)**



# TP- and PT-handles

**Definition:** A transition t and a place p
form a **TP-handle** if there are
two distinct elementary paths $c_1$ and $c_2$ from t to p
such that the only nodes they have in common are t,p

**Definition:** A place p and a transition t
form a **PT-handle** if there are
two distinct elementary paths $c_1$ and $c_2$ from p to t
such that the only nodes they have in common are p,t

# Well-Structured Nets

A net is **well-handled** iff it has:
no PT-handles and no TP-handles

**Definition:** A net is **well-handled** iff
for any pair of nodes x and y of different kinds
(one place and one transition)
any two elementary paths $c_1$ and $c_2$ from x to y
coincide or have some other nodes in common apart x,y

**Definition:** A workflow net N is said **well-structured**
if N* is well-handled

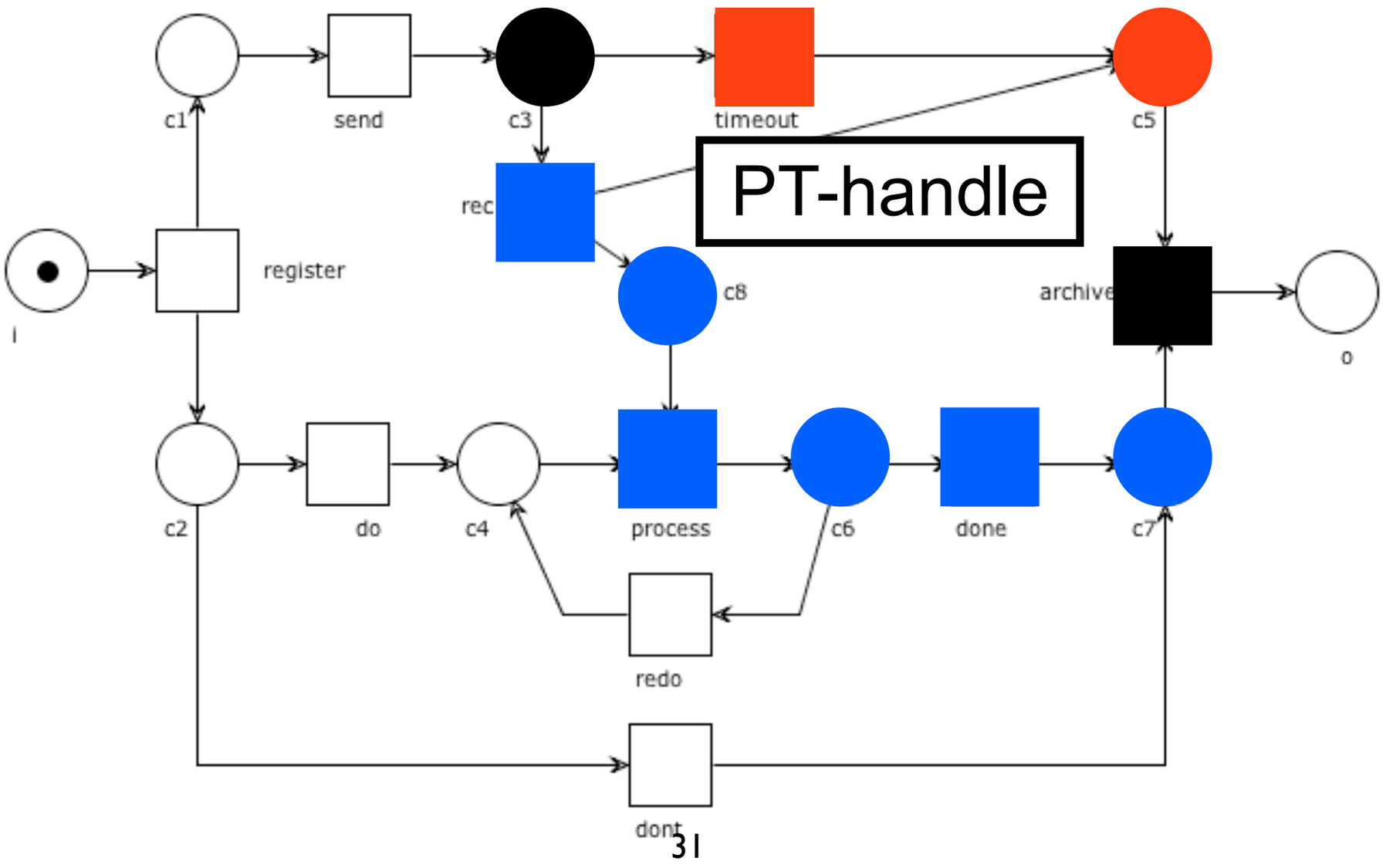
# S-coverability diagnosis

**Theorem:**
If N is sound and well-structured, then N* is S-coverable
(proof omitted)

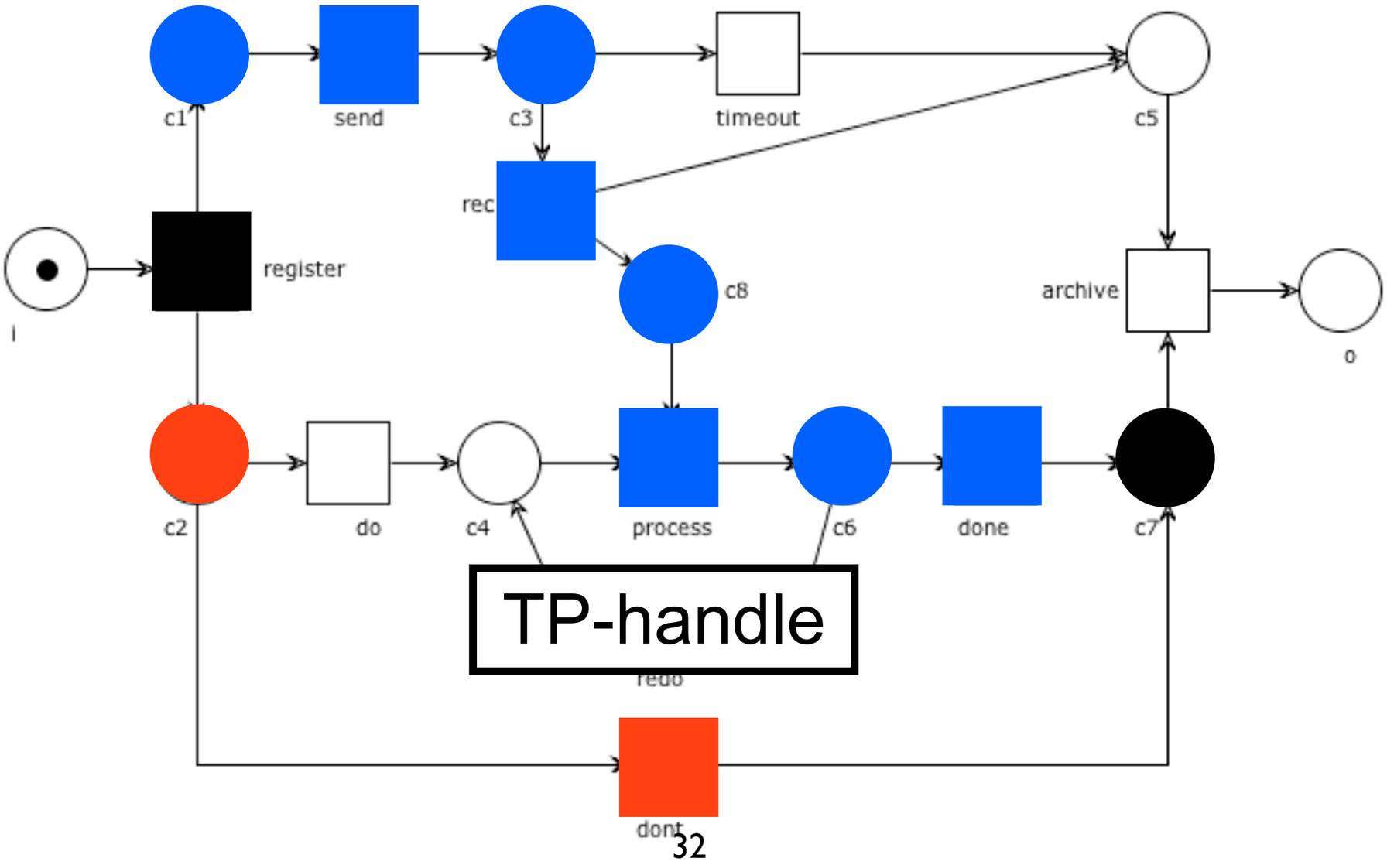
**N well-structured + N* not S-coverable => N not sound**

Note that
**If N* is not well-handled, N can be sound**
especially if reset is involved in the handle
(it is a symptom, not a disease)

# Running example: Well-structured? No



# Running example: Well-structured? No

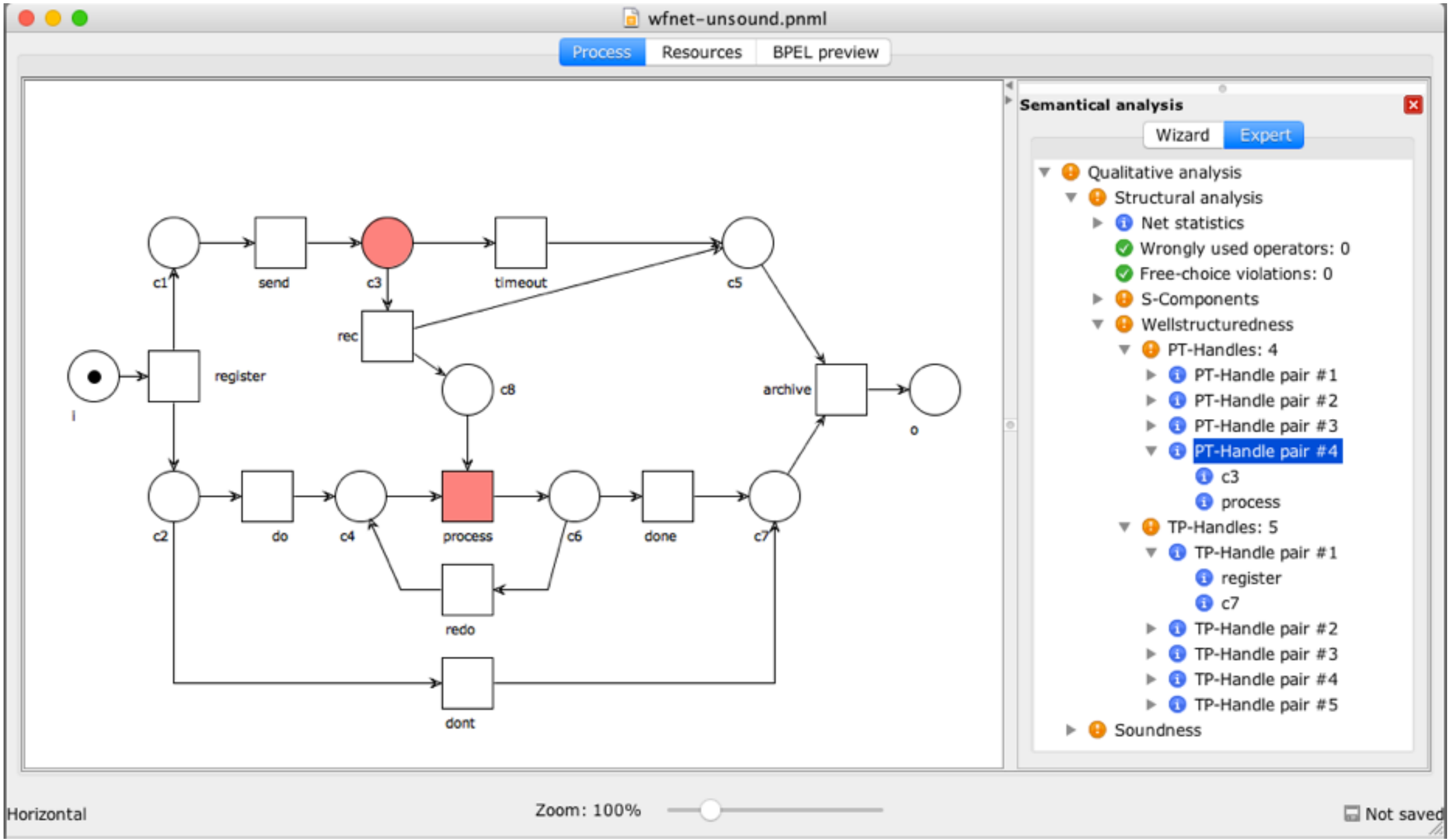


# Be careful

N well-structured = N* well-handled

WoPeD marks PT/TP-handles over N*
(not over N)

# Running Example: WoPeD Diagnosis



# Liveness and boundedness

# vs

# Soundness requirements

# Improper completion

Suppose **N completes improperly:**
from i we can reach o+M

We can do the same on N*
then we fire reset and reach i+M

we can repeat the same run and reach i+2M
and then i+3M and then i+4M and then ... i+kM

then **N* has some unbounded places**
(all p such that M(p)>0)

# Unsoundness from unboundedness

**Improper completion of N** implies **unboundeness of N***

**Symptom: N* has some unbounded places**

**Disease: N could complete improperly**

# Consequences of boundedness

If N* is bounded, then:
if o+M is reachable from i in N, then M=0

**If N* is bounded, then**

either N satisfies

**both option to complete and proper completion**

or N does not satisfy **option to complete**

# Completion option failure

**Suppose N does not satisfy the "option to complete":**
then from i we can reach M
from which we cannot mark o

We can do the same on N*
then reset is dead from M
i.e. reset is non-live in N*

**N* has non-live transitions (including reset)**

# Unsoundness from non-liveness

**Option to complete fail for N** implies **non-liveness of N***

**Symptom: reset transition is non-live in N***

**Disease: N could violate option to complete**

# Unsoundness from Non-Liveness

If N* **is bounded and has dead transitions**, then

if reset is dead
N and N* have the same finite reachability graph
**hence N has the same dead tasks as N***
(except reset)

if reset is not dead
the reachability graphs of N and N* differ only for $o \xrightarrow{reset} i$
(because N* is bounded)
**hence N has the same dead tasks as N***

# Unsoundness from Non-Liveness

**Symptom: N* is bounded and has dead transitions**

**Disease: N has the same dead tasks as N***

# Unsoundness from Non-Liveness

**Symptom: N* has non-live transitions**

**Disease: N could have dead transitions**

(but which ones?)

# Error sequences

# Diagnostic information

The sets of:
unbounded places of N*
dead transitions of N*
non-live transitions of N*

may provide useful information for
the diagnosis of behavioural errors
(pointing to different types of errors)

Unfortunately, this information is not always sufficient
to determine the exact cause of the error

**Behavioural error sequences** can overcome this problem

# Error sequences

Rationale:
We want to find firing sequences such that:

every continuation of such sequences will lead to an error

they have minimal length
(none of its prefixes satisfies the above property)

Informally:
error sequences are scenarios that capture
the essence of errors made in the workflow design
(violate “option to complete” or “proper completion”)

# Non-Live sequences: informally

A **non-live sequence** is a
firing sequence of **minimal length**
such that **completion of the case is no longer possible**

i.e. a witness for transition reset being non-live in N*

# Non-Live sequences: fundamental property

Let N be such that:
N* is bounded
N (or equivalently N*) has no dead task

Then, N* is live
iff
N has no non-live sequences

# Non-Live sequences: graphically

**The analysis is possible in bounded systems only**

Compute the RG of N*

Color in **red** all nodes from which there is **no path** to o

Color in **green** all nodes from which **all paths** lead to o

Color in yellow all remaining nodes
(some but not all paths lead to o)

# Non-Live sequences: remarks

No **red** node implies no yellow node

No **green** node implies no yellow node

# Non-Live sequences: formally

**Definition**:

An occurrence sequence

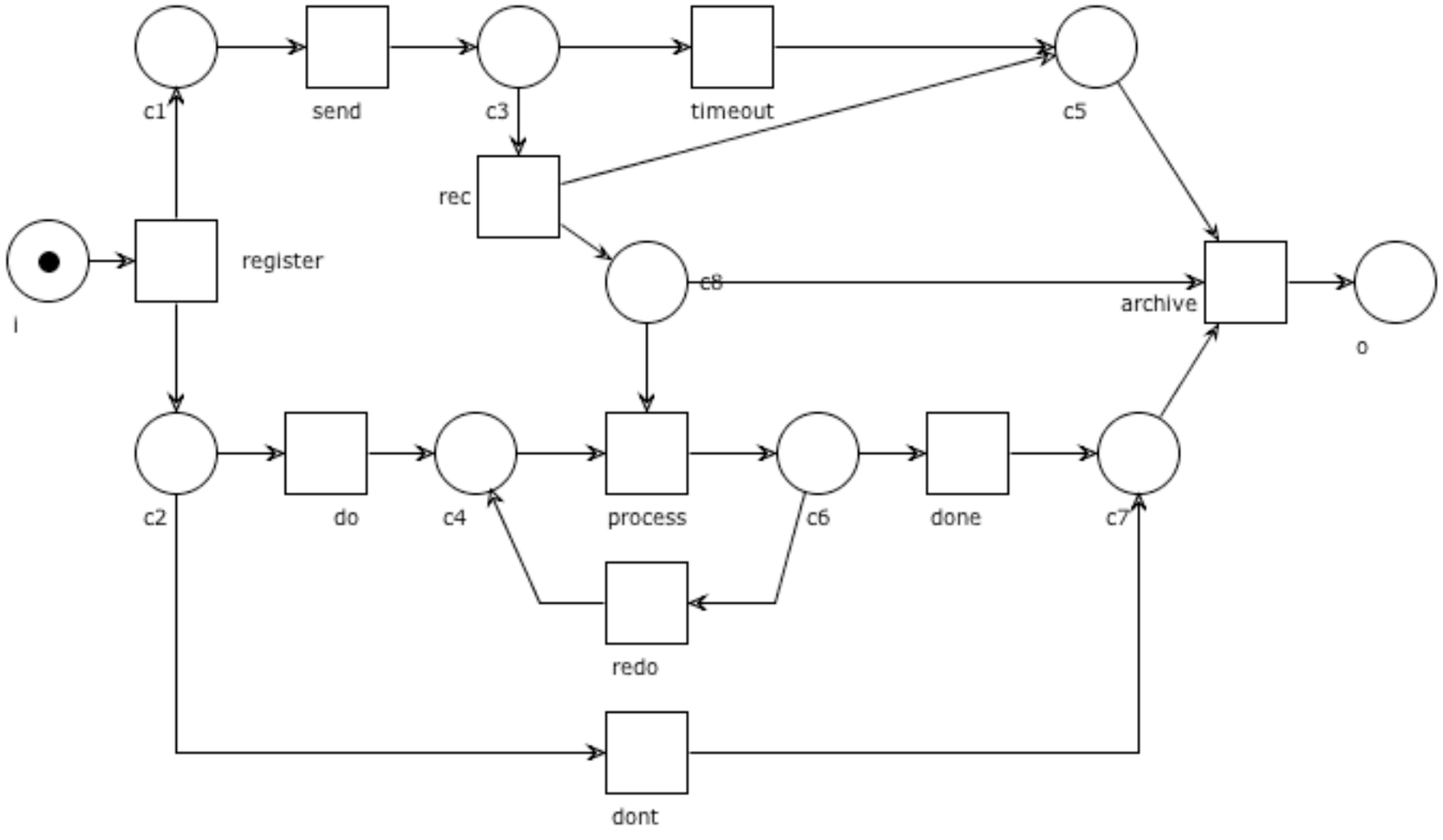
$i \xrightarrow{t_1} M_1 \ldots M_{k-1} \xrightarrow{t_k} M_k$ is **non-live** if

- all markings are distinct
- $M_{k-1}$ is yellow
- $M_k$ is red

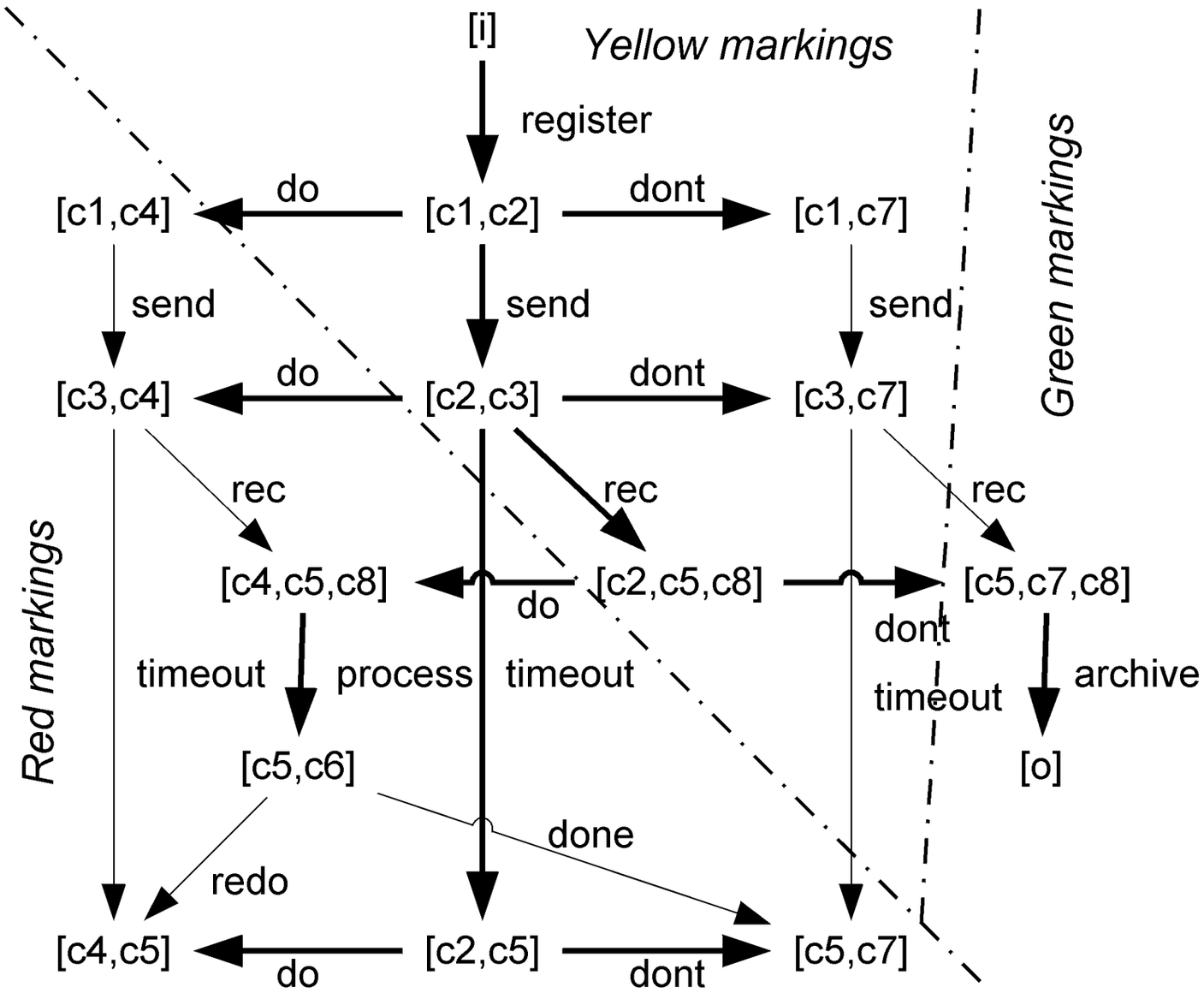
Firing $t_k$ removes the option to complete!

Then, the firing sequence $t_1 \ldots t_k$ is also called **non-live**

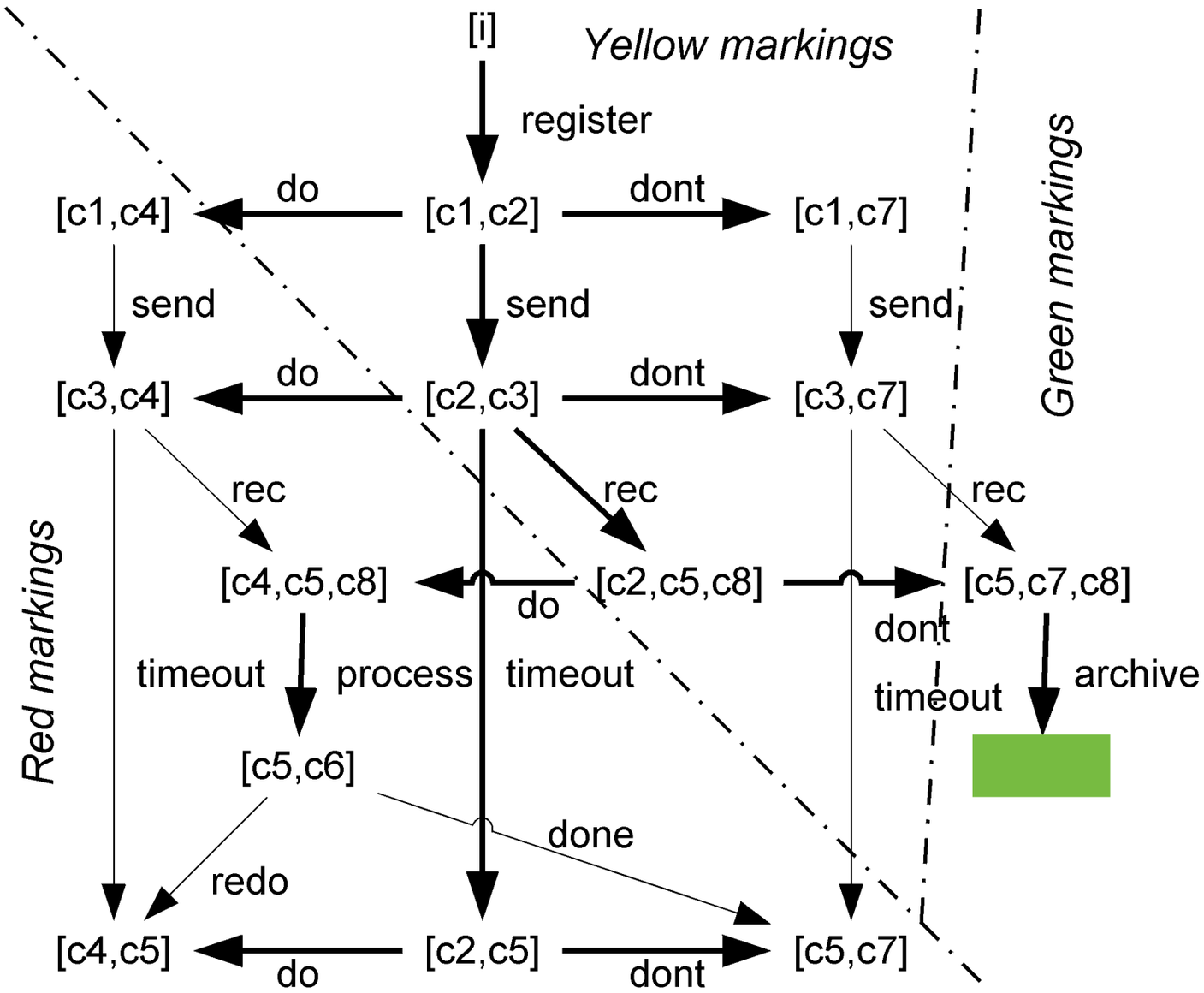
# Running example: slight variant



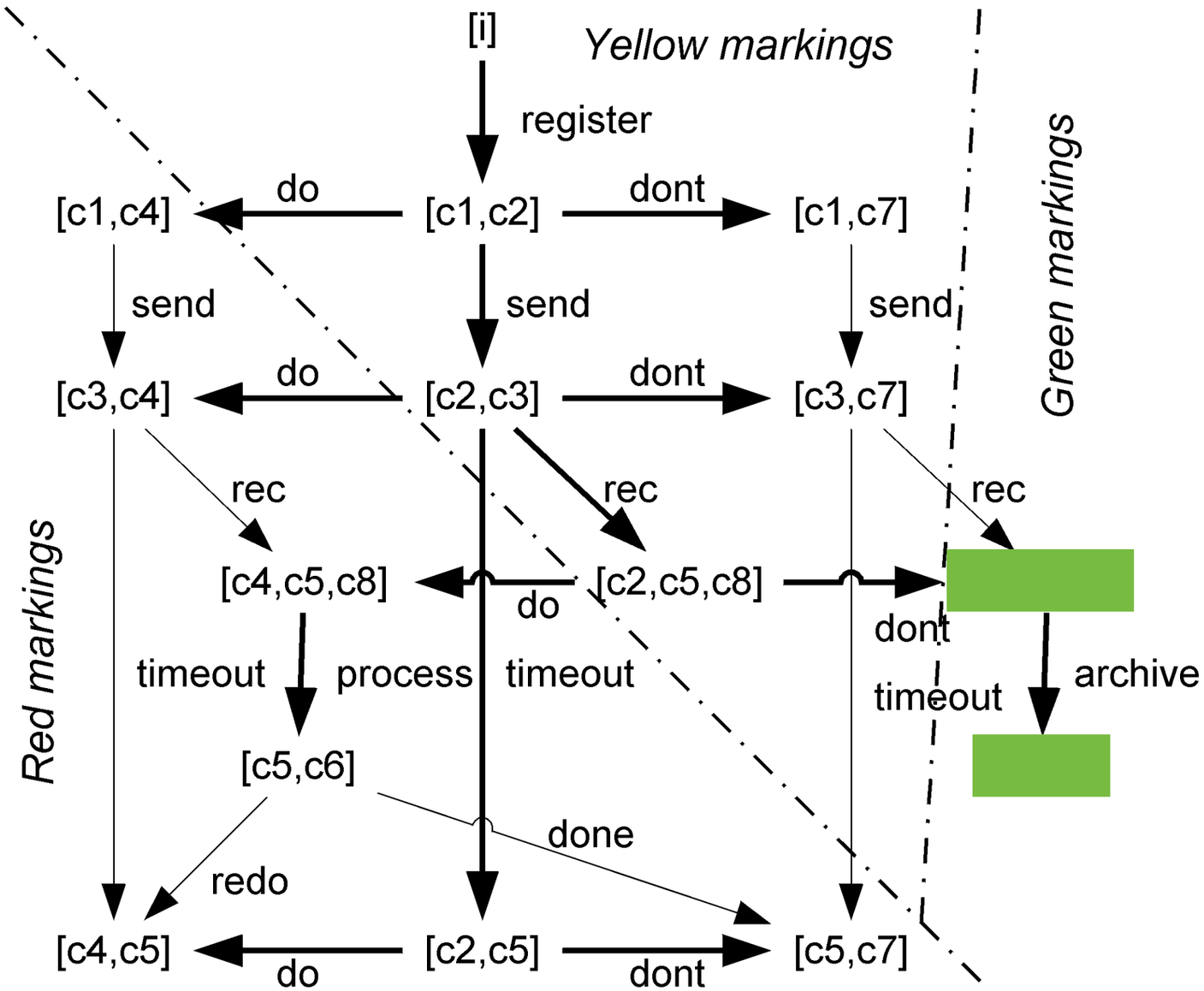
# Running example variant: colored RG



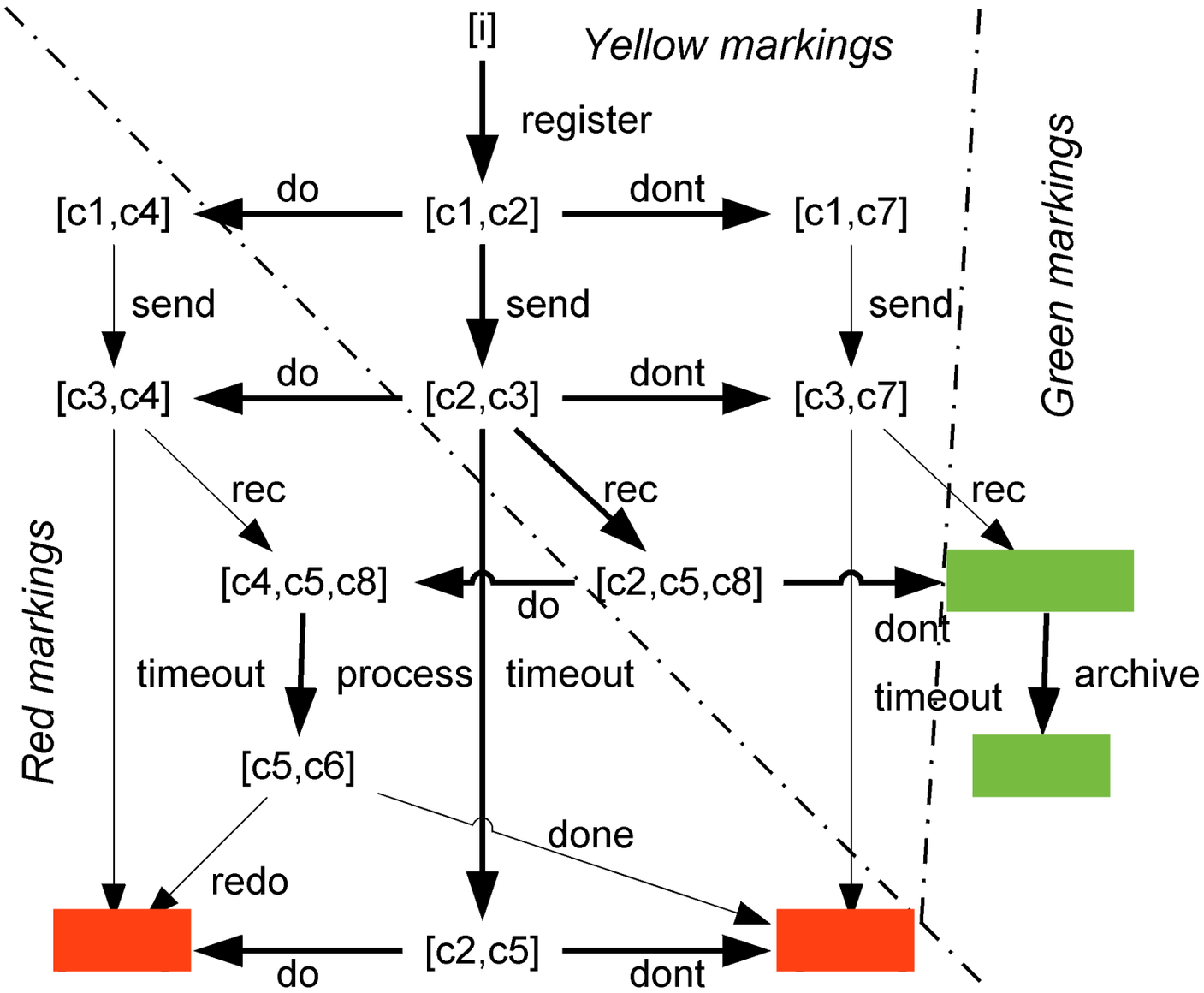
# Running example variant: colored RG



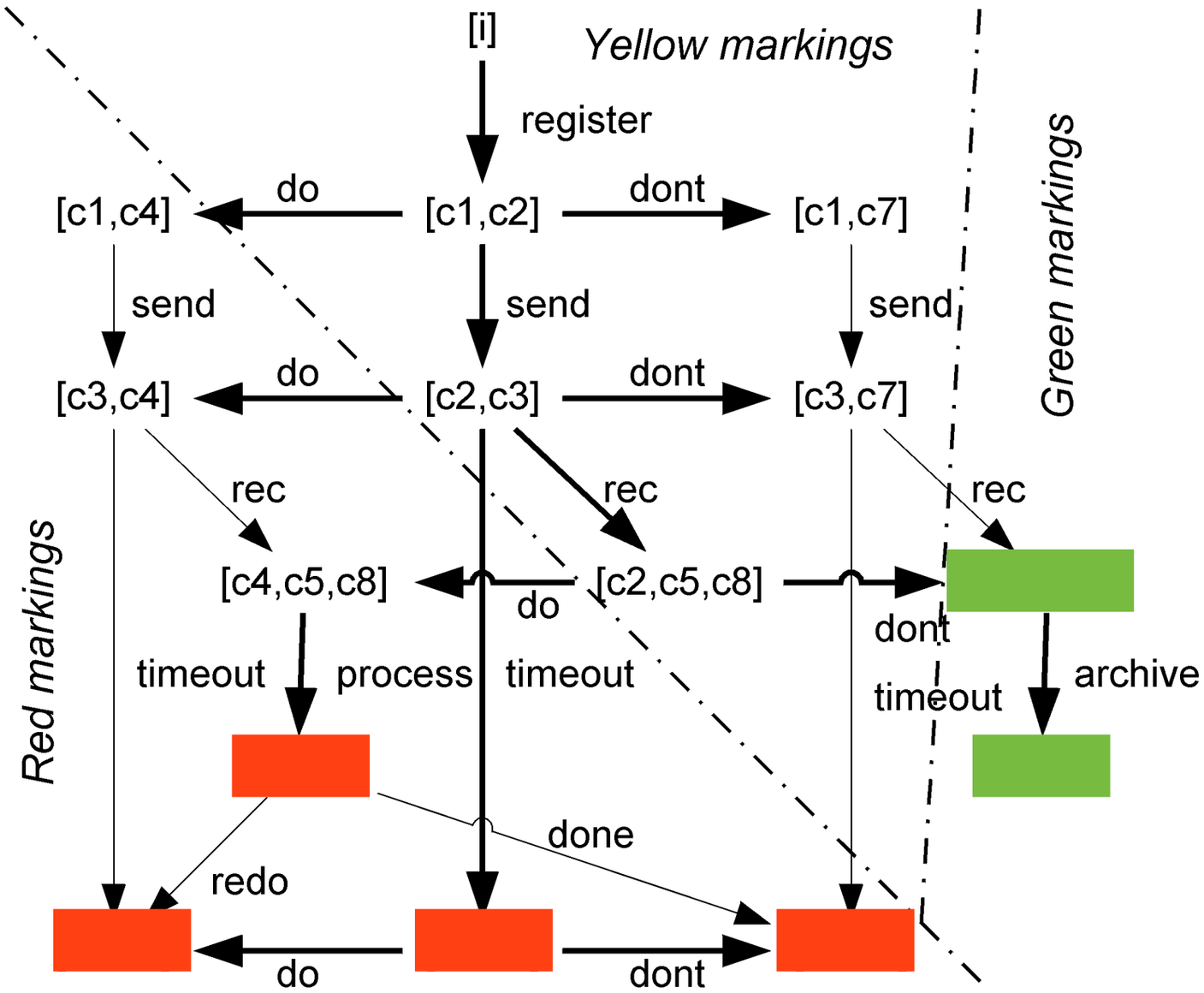
# Running example variant: colored RG



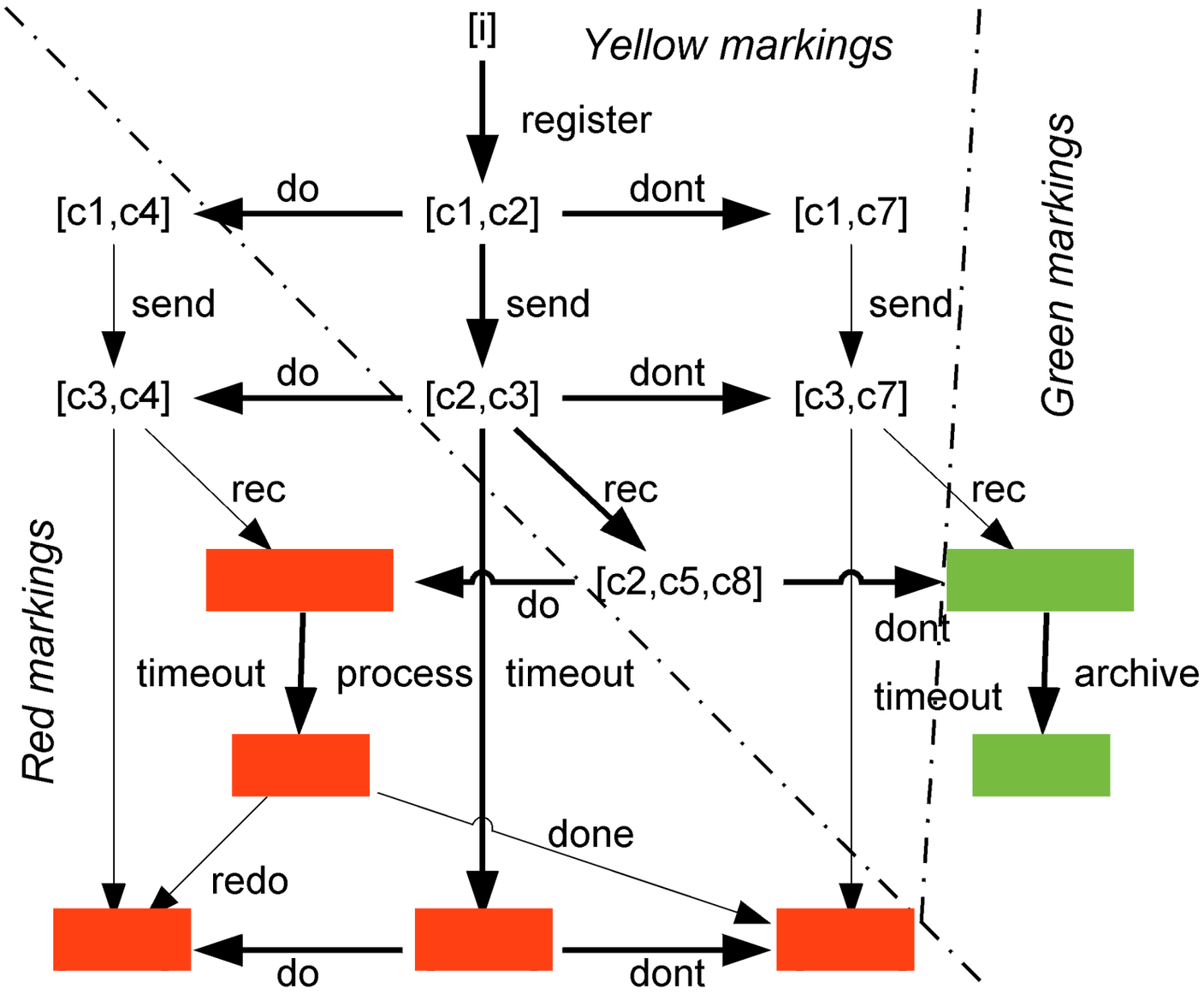
# Running example variant: colored RG



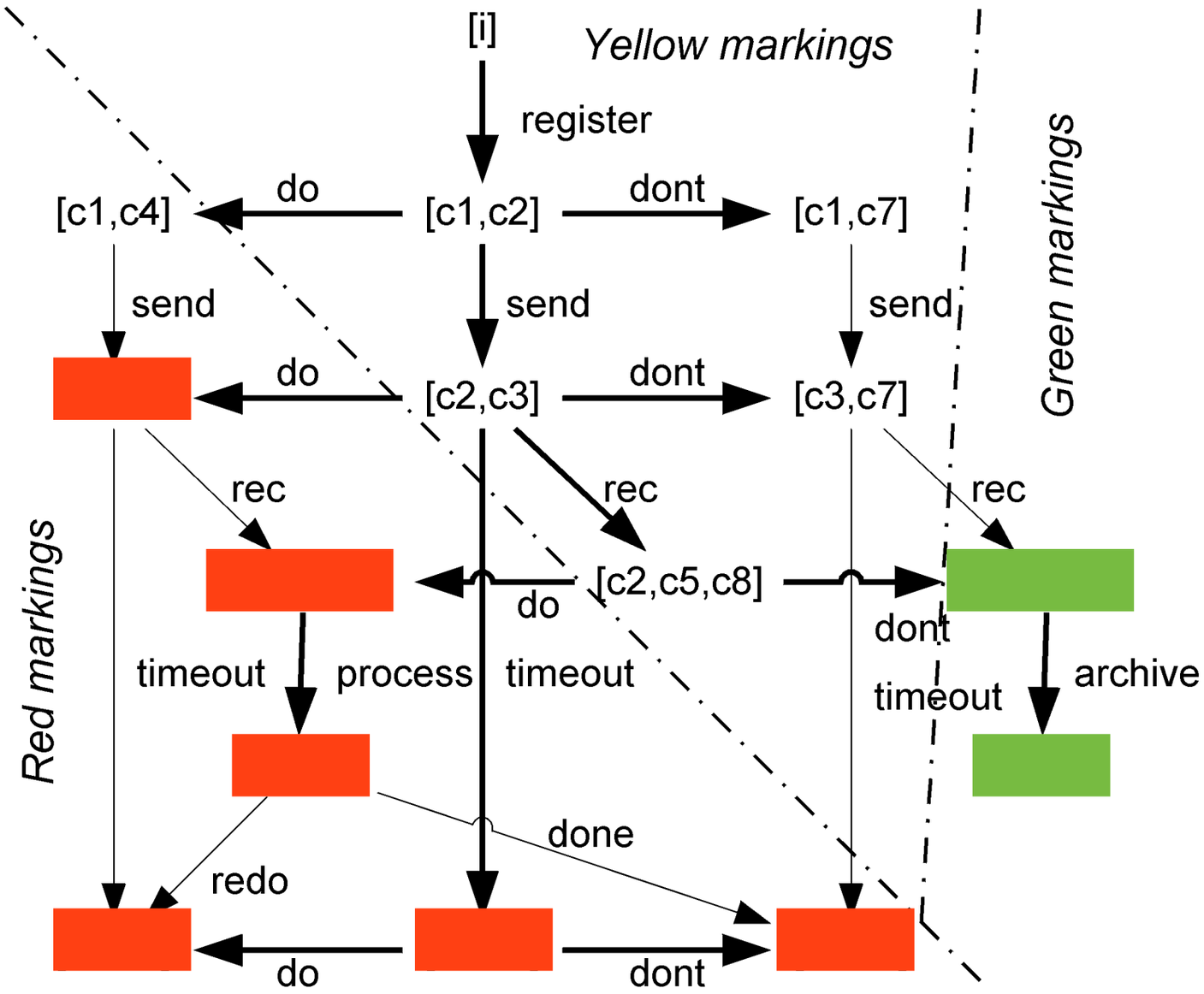
# Running example variant: colored RG



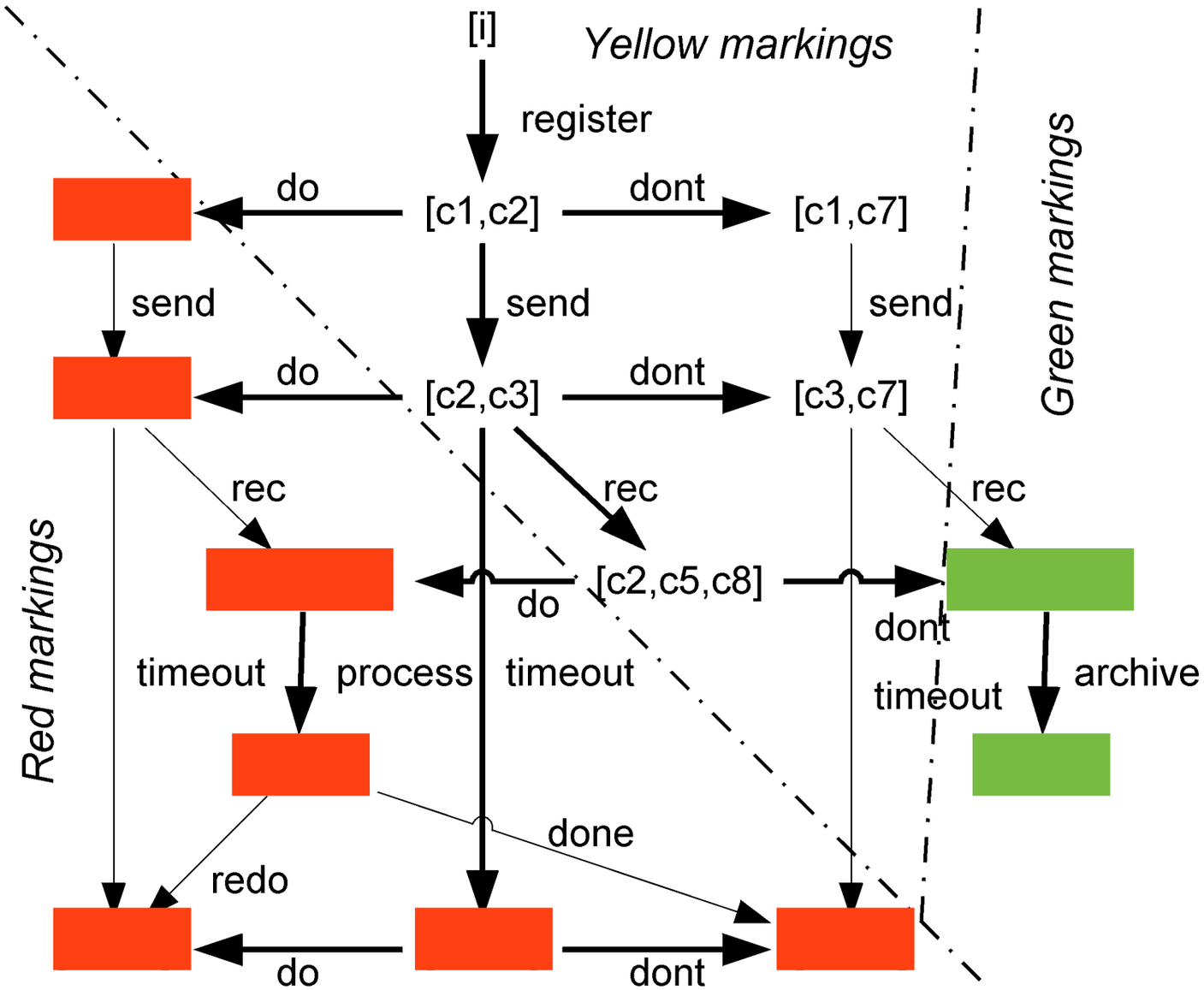
# Running example variant: colored RG



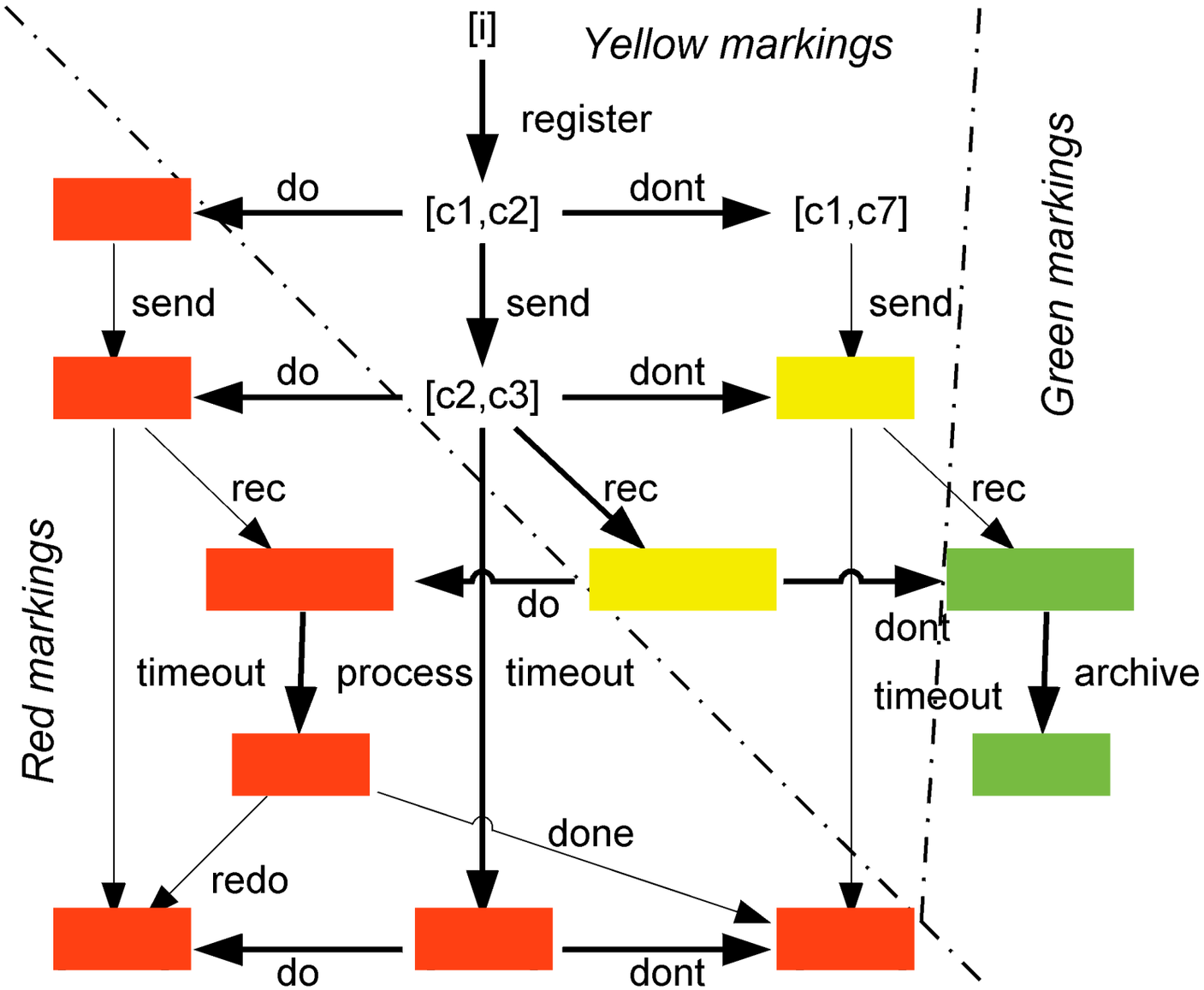
# Running example variant: colored RG



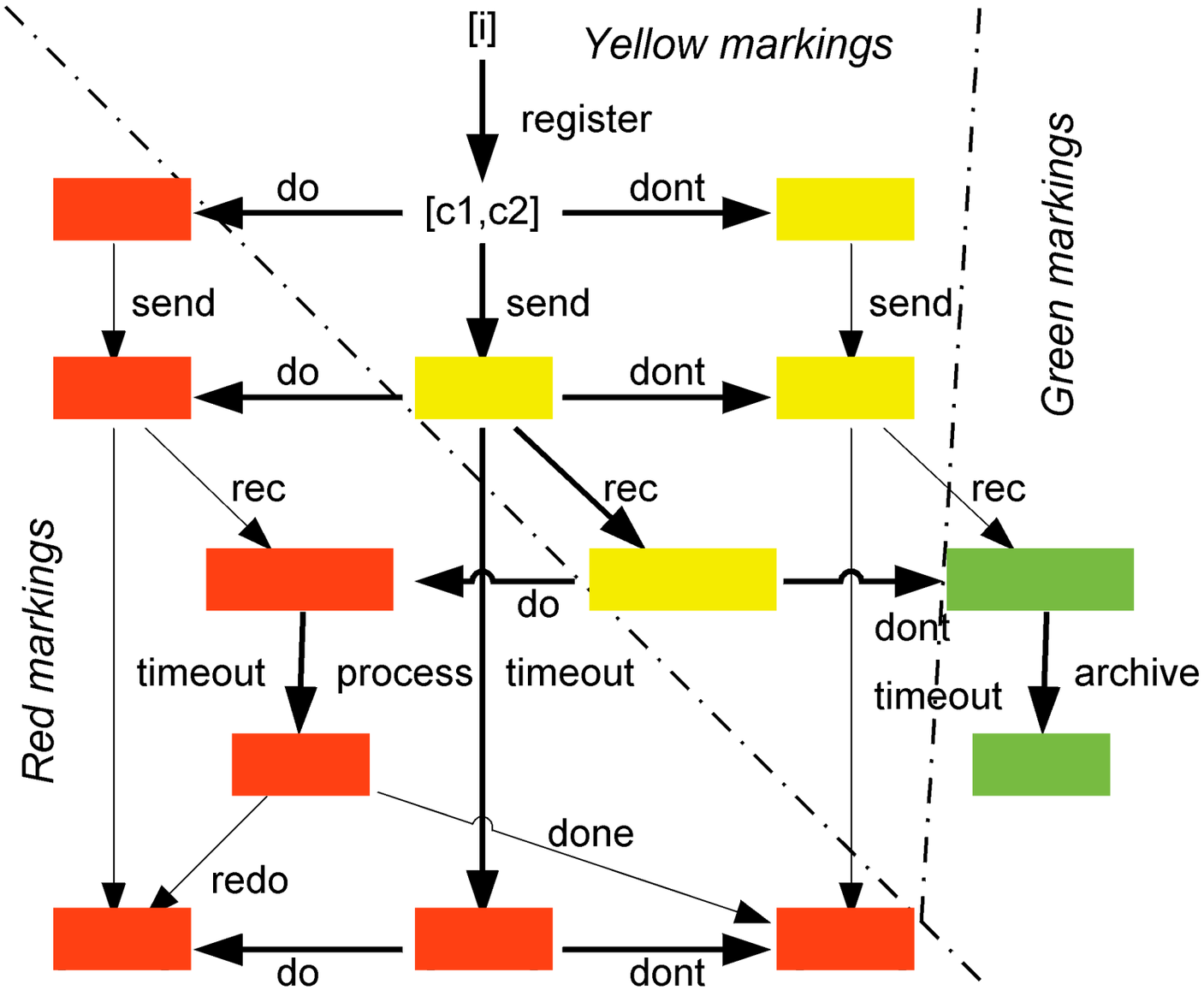
# Running example variant: colored RG



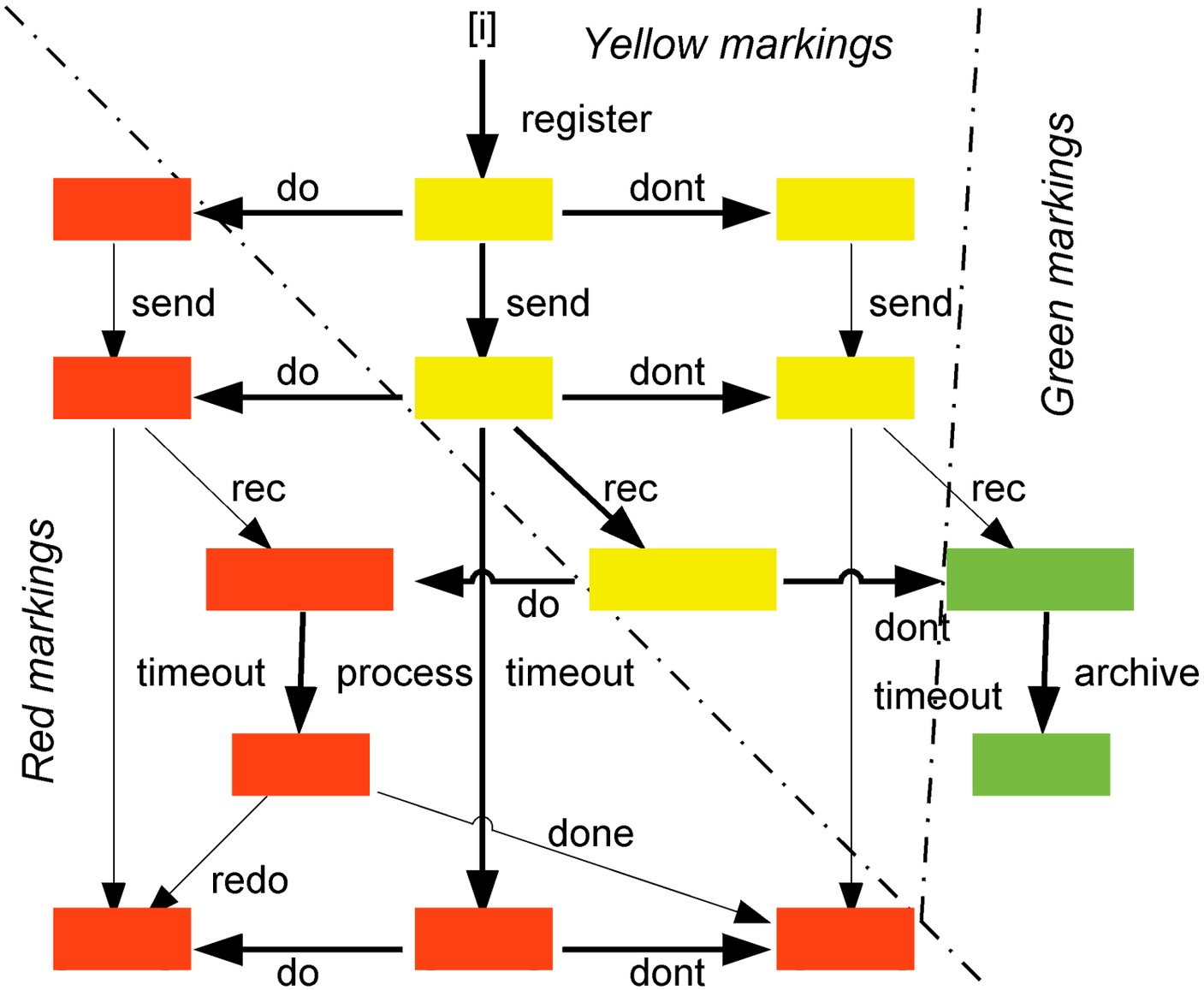
# Running example variant: colored RG



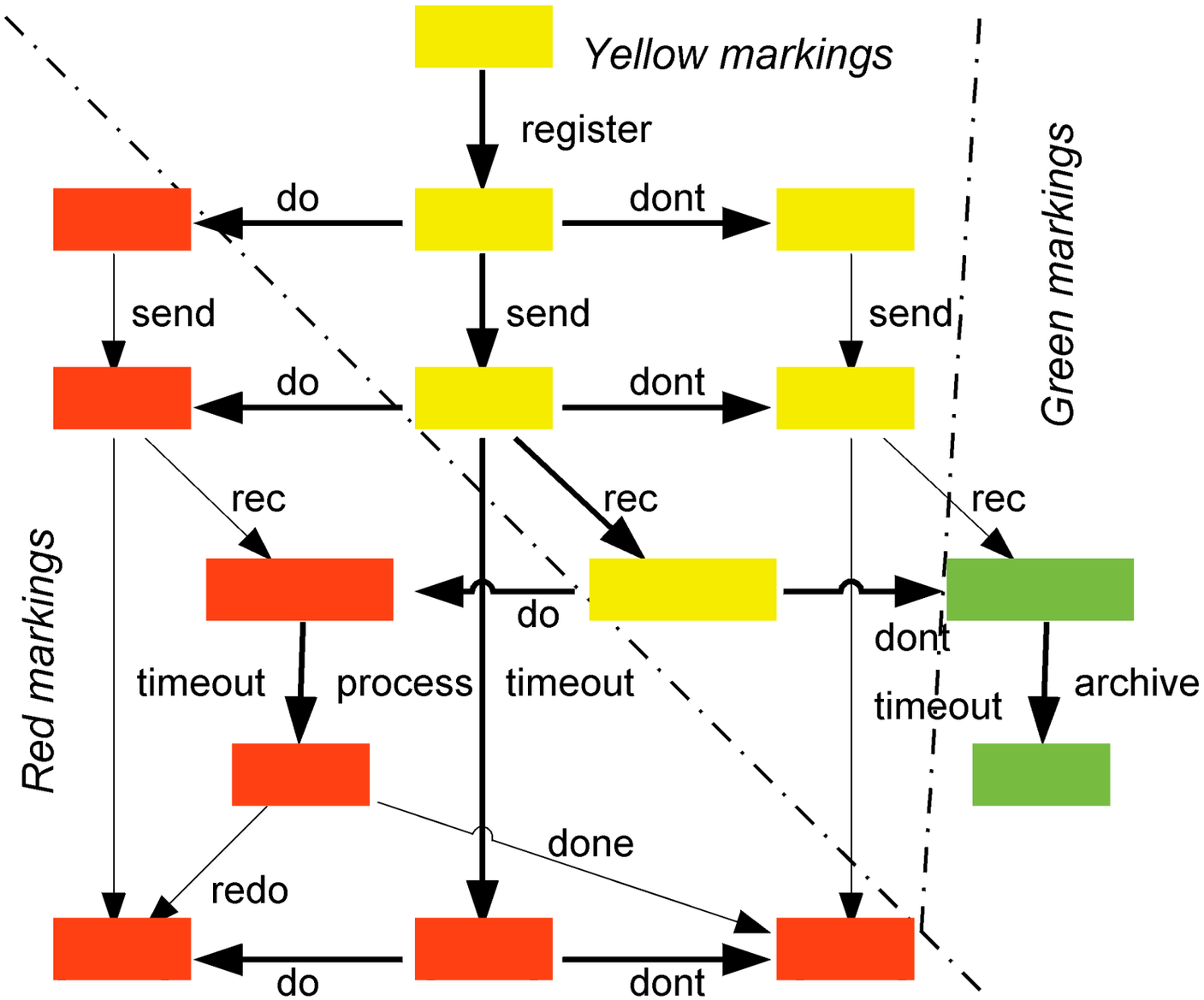
# Running example variant: colored RG



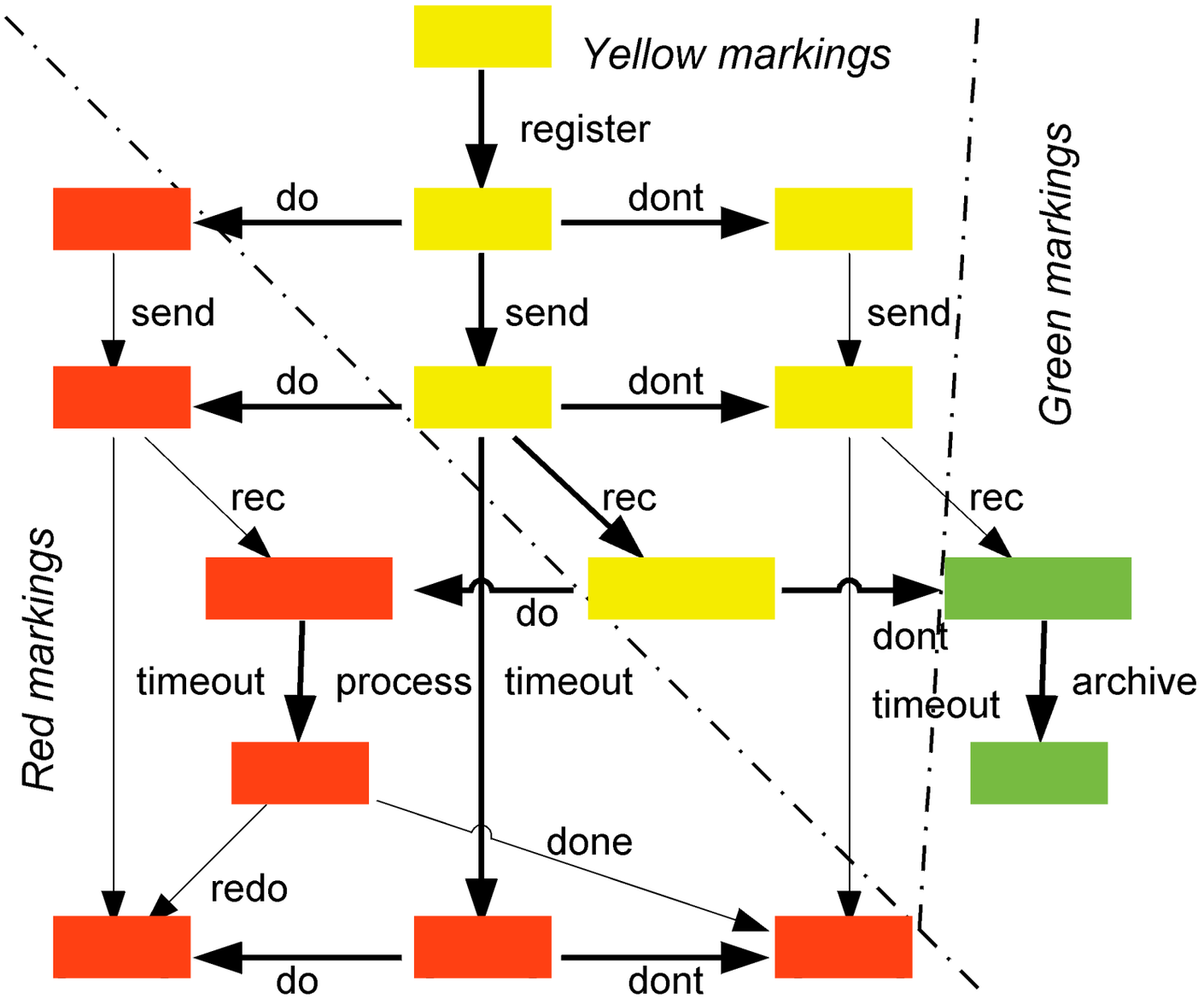
# Running example variant: colored RG



# Running example variant: colored RG



# Running example variant: colored RG




register, do

register, send, do

register, send, timeout

register, send, rec, do

register, send, dont, timeout

and also?

# Unbounded sequences: informally

An **unbounded sequence** is a
firing sequence of **minimal length** such that
every continuation leads to **invalidate proper completion**

i.e. a witness for unboundedness

# Unbounded sequences: fundamental property

N* is bounded
iff
N has no unbounded sequences

Undesired markings:
**infinite-weighted markings or markings greater than o**

# Unbounded sequences: graphically

Compute the CG of N*

Color in **green** all nodes from which
**undesired markings are not reachable**

Color in **red** all nodes from which
**no green marking is reachable**
(undesired markings are unavoidable)

Color in yellow all remaining nodes
(undesired markings are reachable but avoidable)

# Unbounded sequences: remarks

No **red** node implies no yellow node

No **green** node implies no yellow node

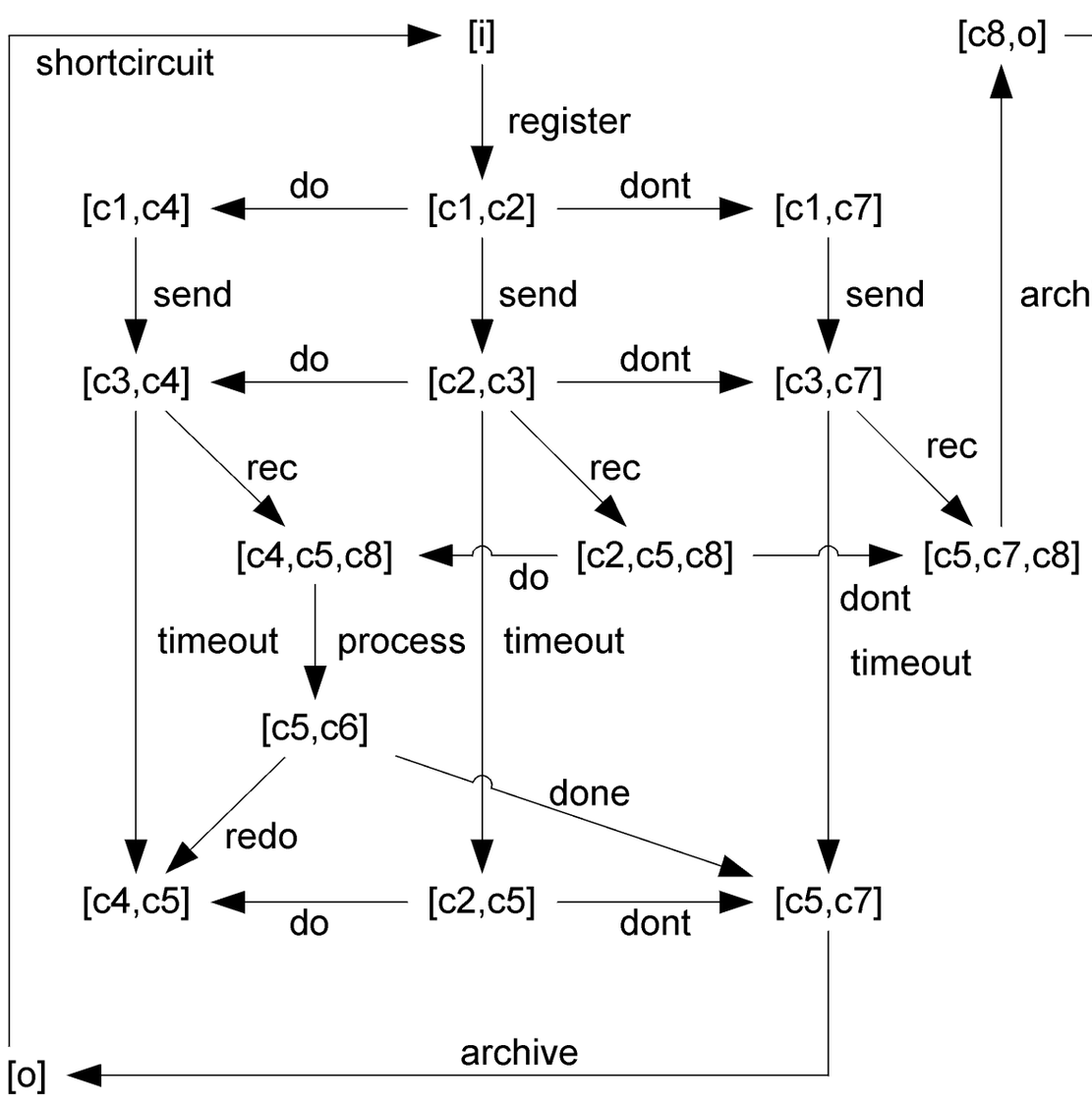
# Restricted coverability graph (RCG)

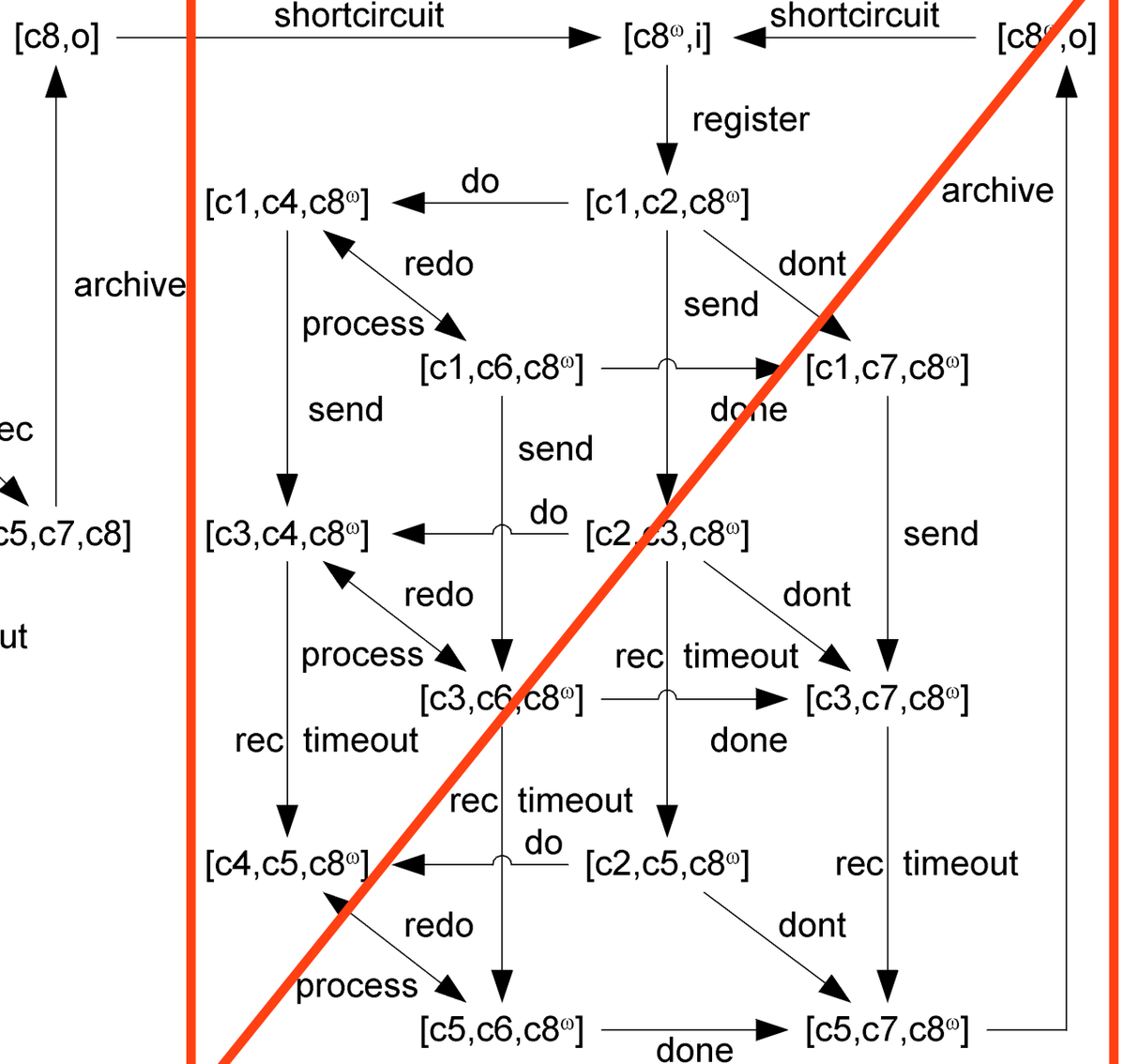
CG can become very large (intractable!)

Basic observation:
infinite-weighted markings leads to infinite-weighted markings
and they will be all red

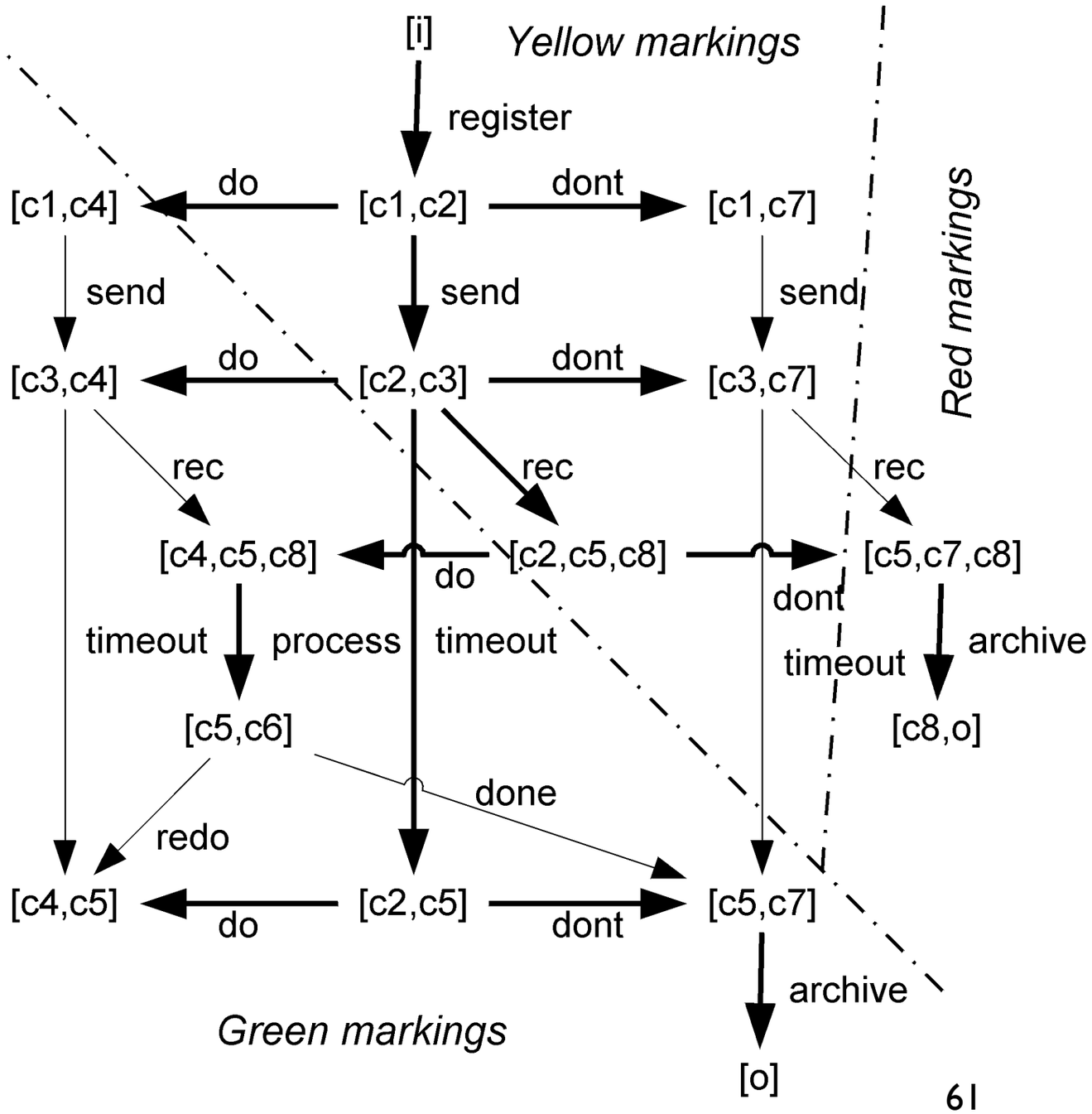
We can just avoid computing them!

# Running example: RCG vs CG

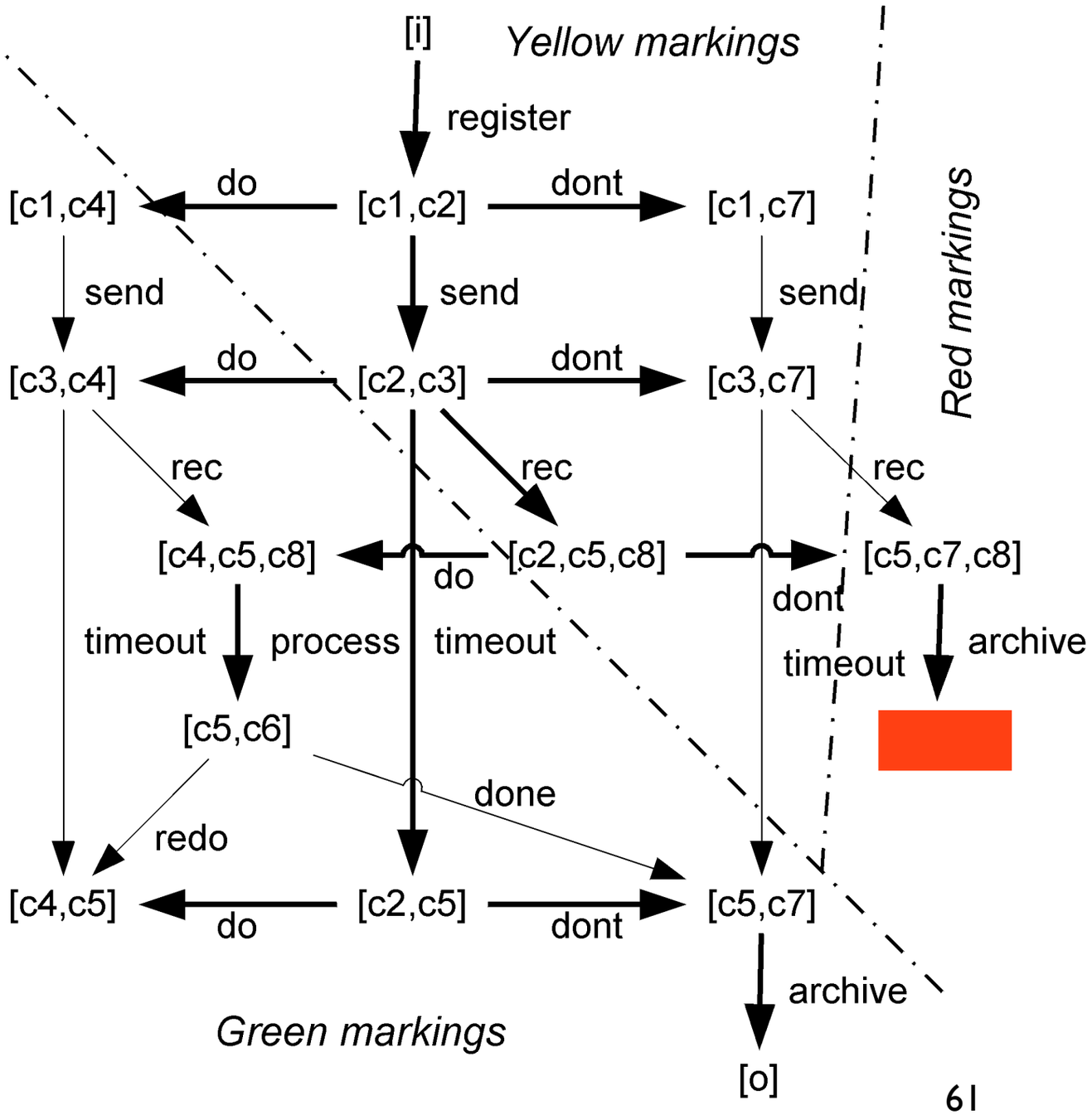


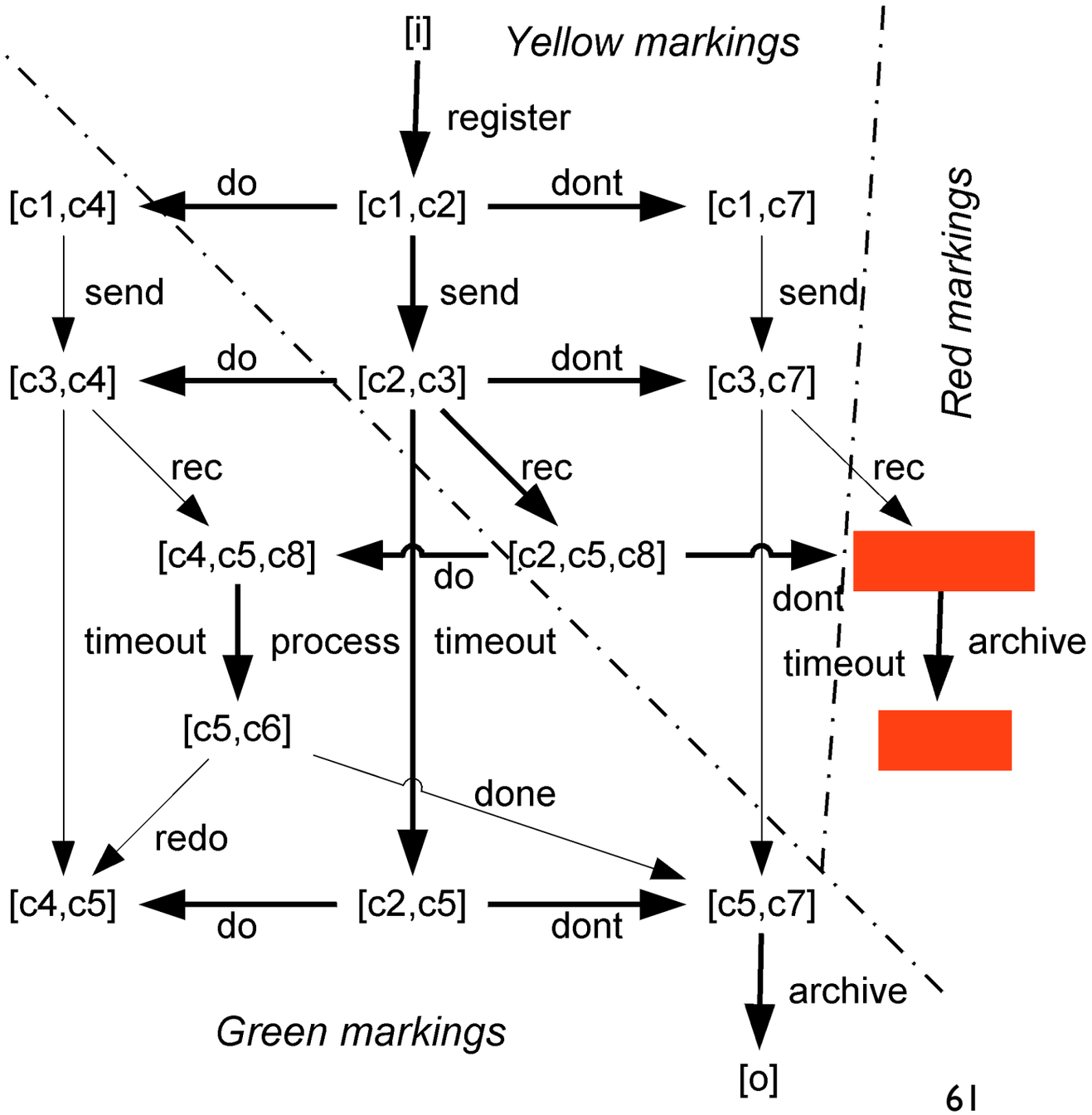
# Running example: colored RCG



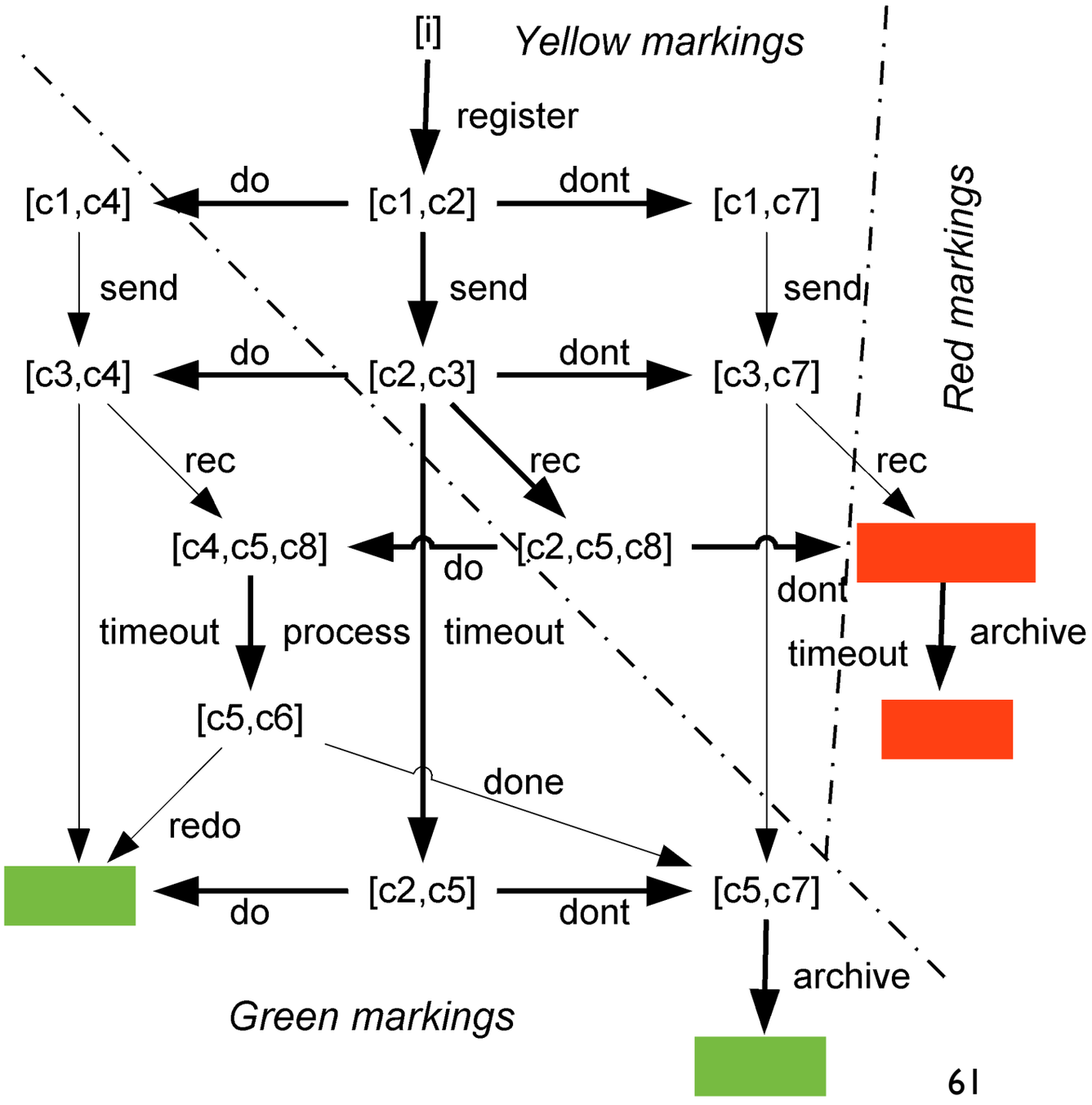
# Running example: colored RCG



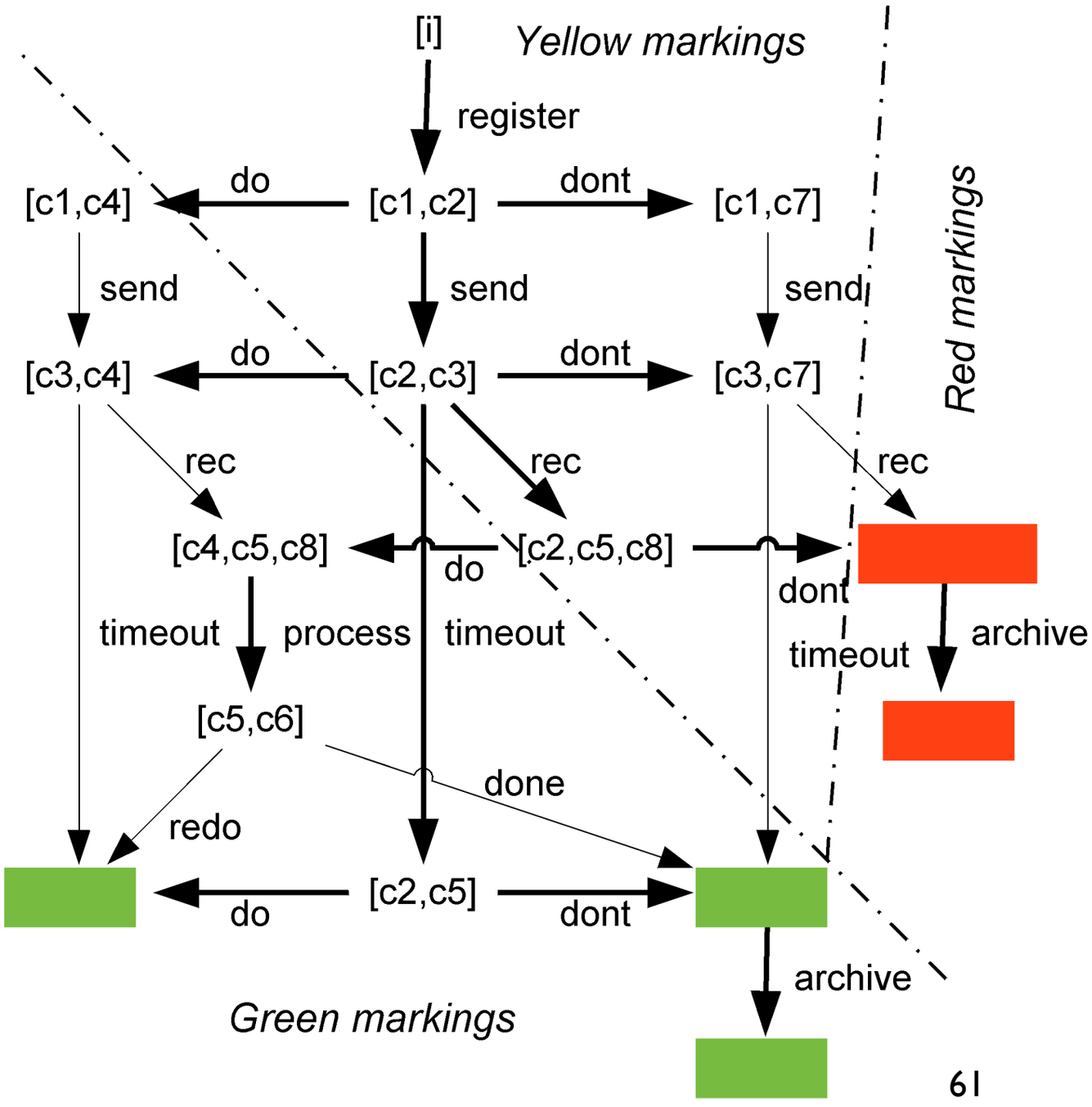
# Running example: colored RCG



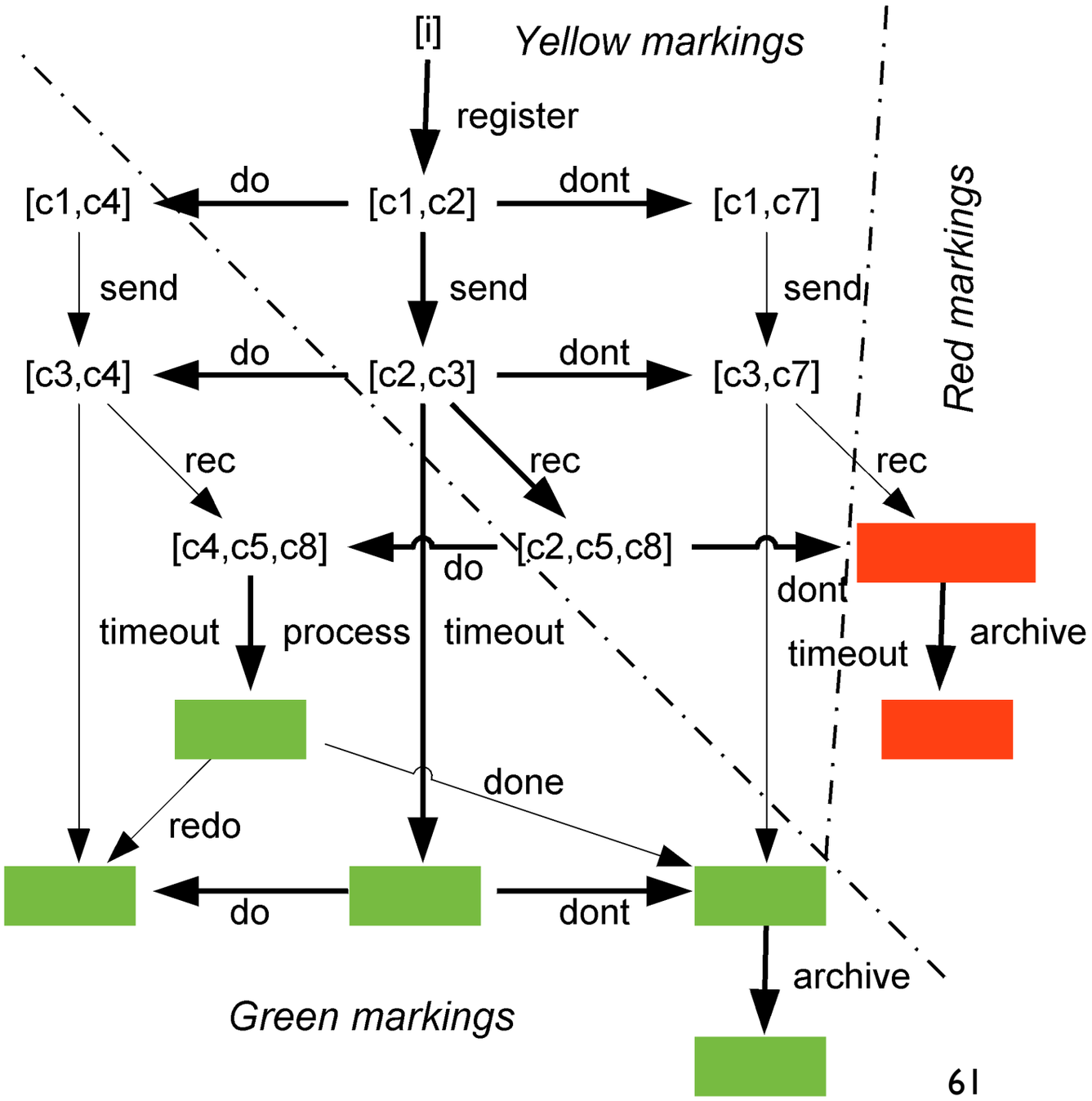
# Running example: colored RCG



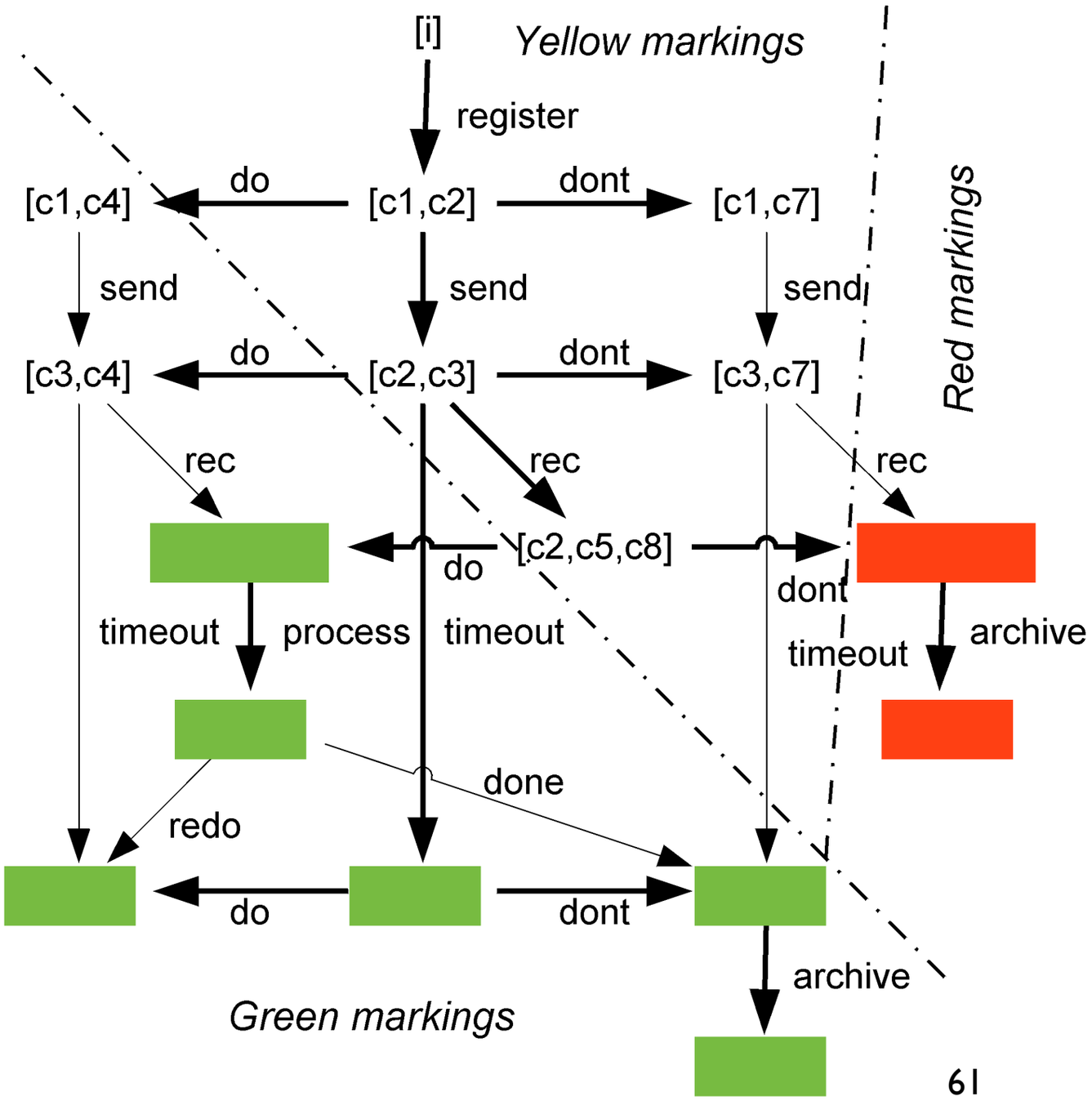
# Running example: colored RCG



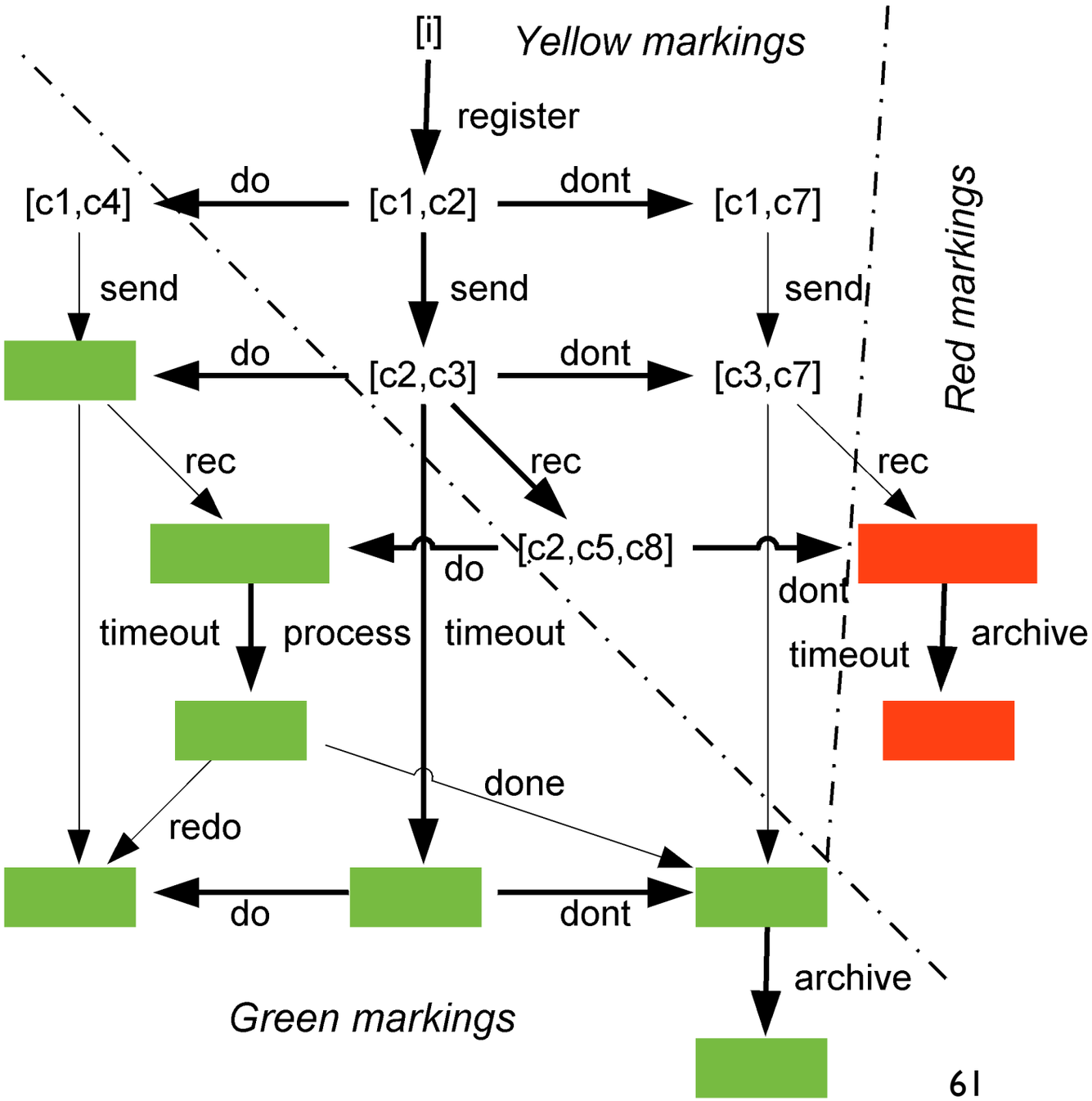
# Running example: colored RCG



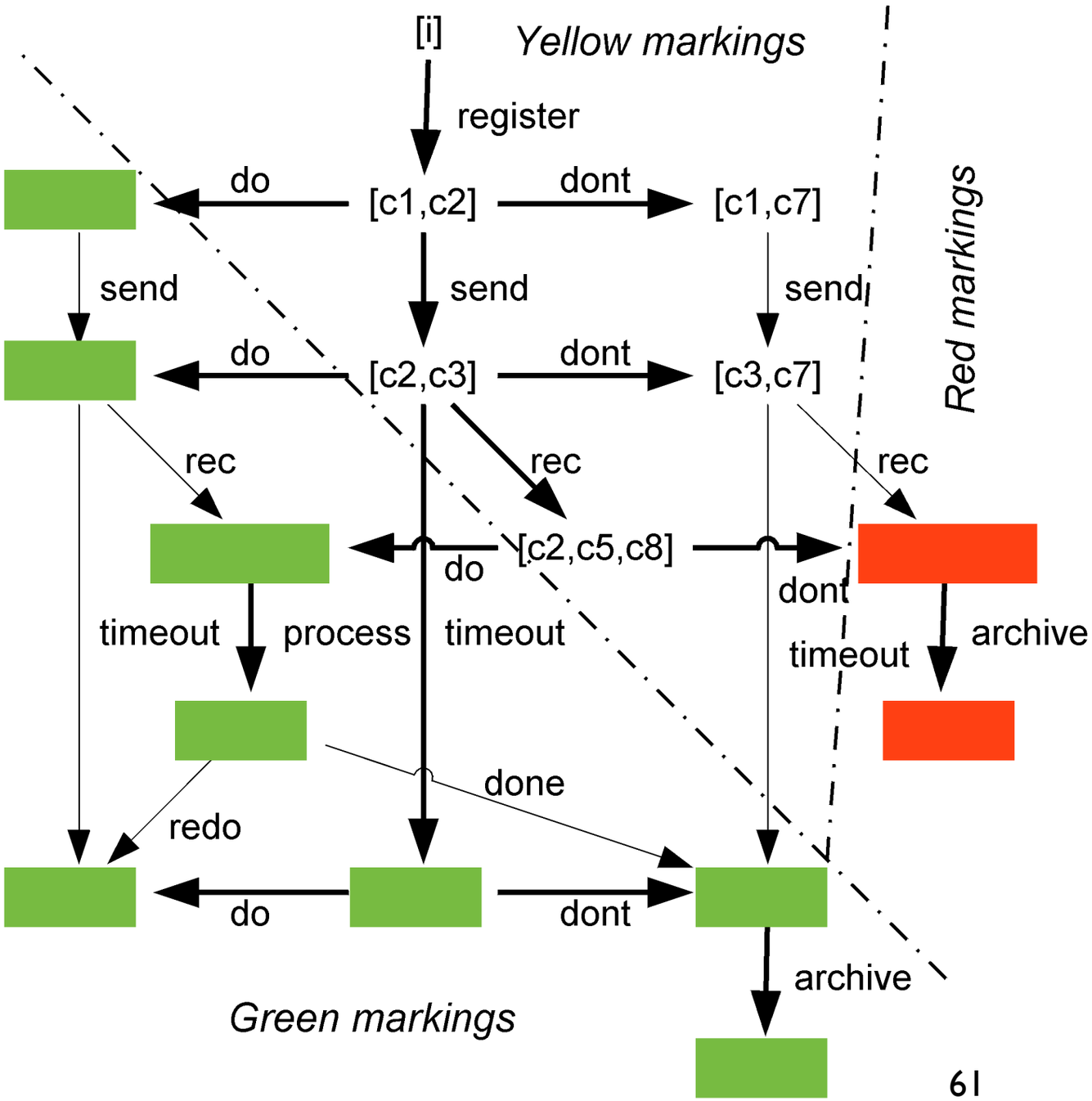
# Running example: colored RCG



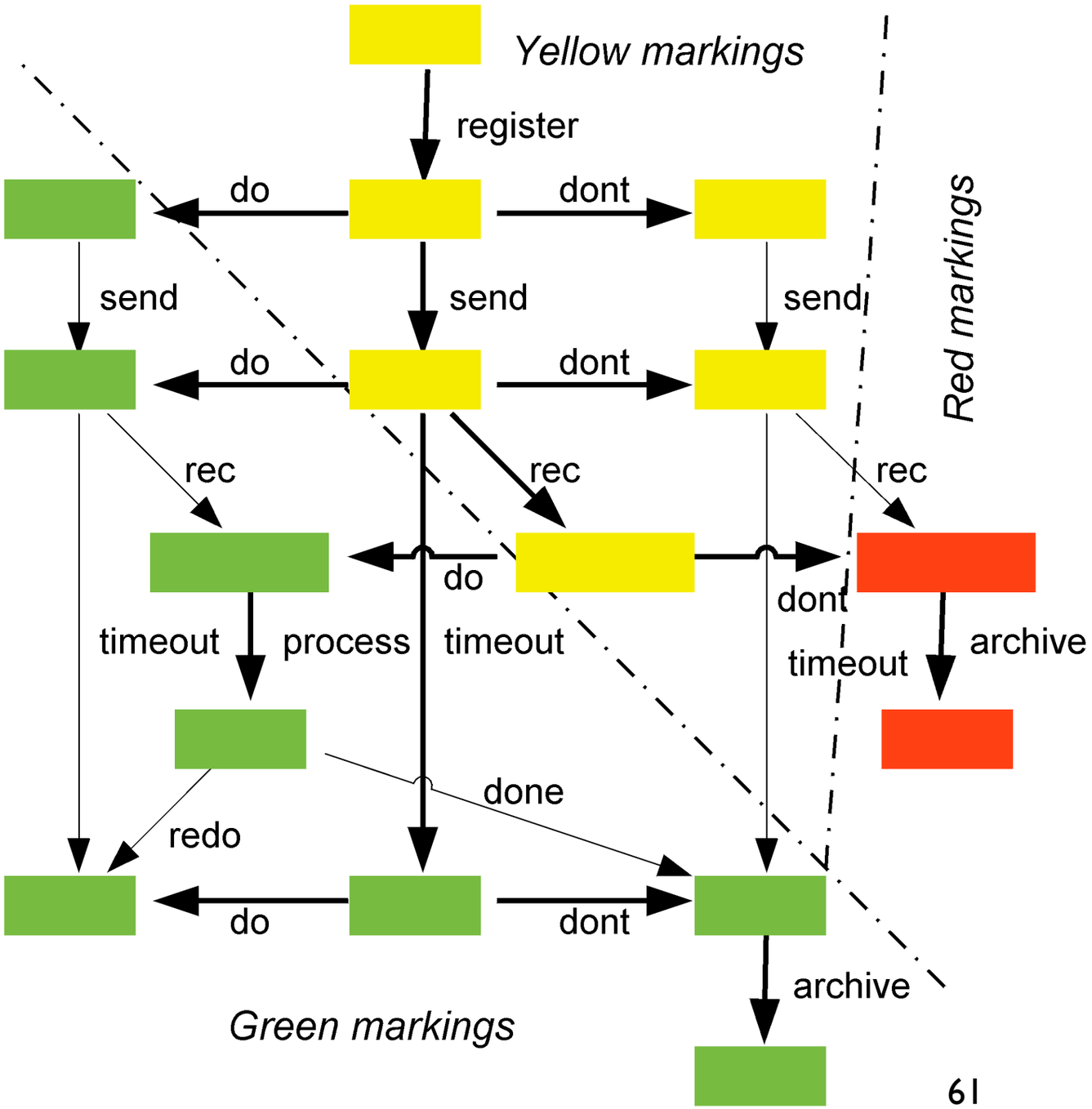
# Running example: colored RCG



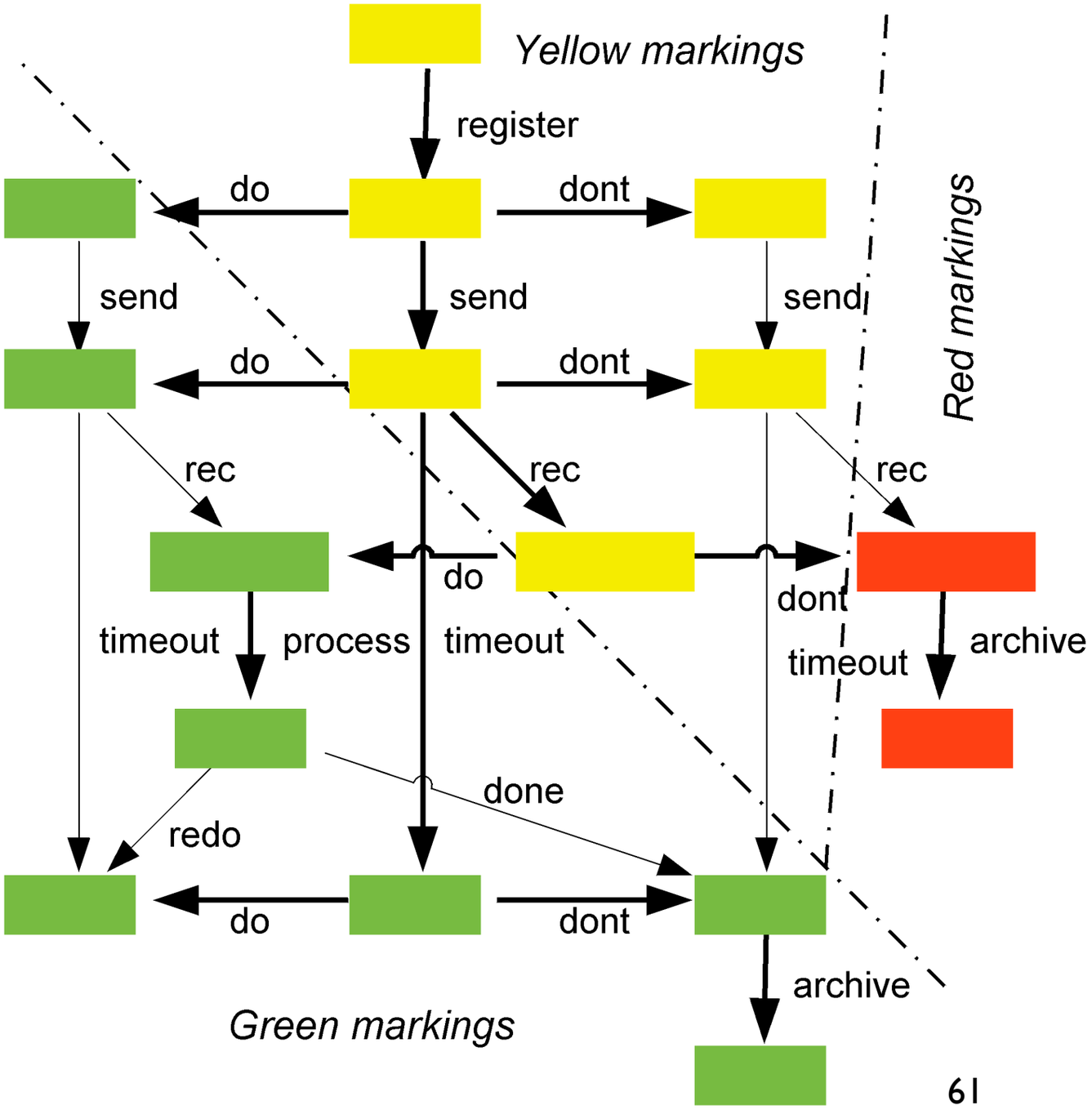
# Running example: colored RCG



# Running example: colored RCG



# Running example: colored RCG



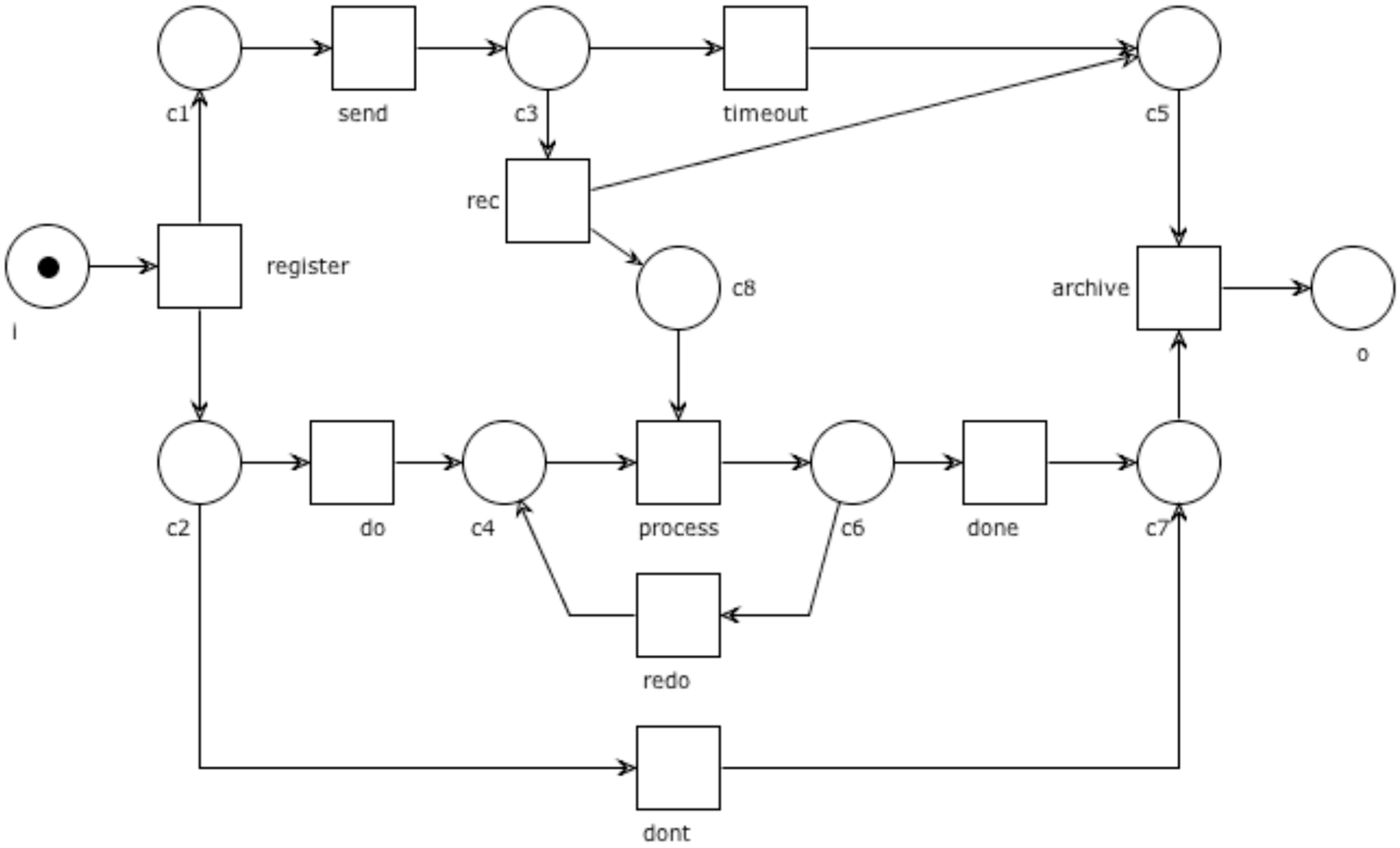

register, dont, send, rec

register, send, dont, rec

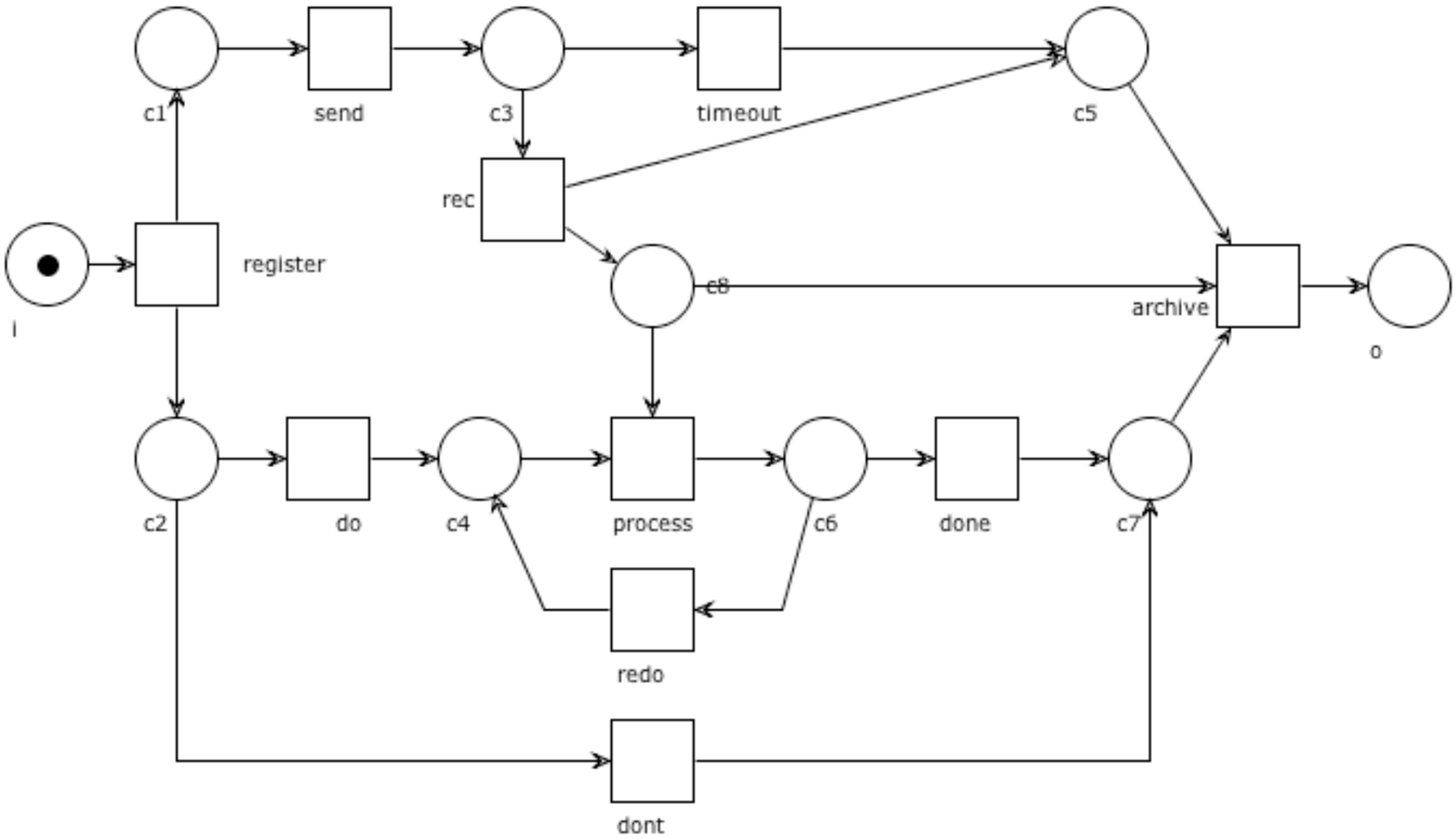
and also?

# Practice with WoPeD (and Woflan)

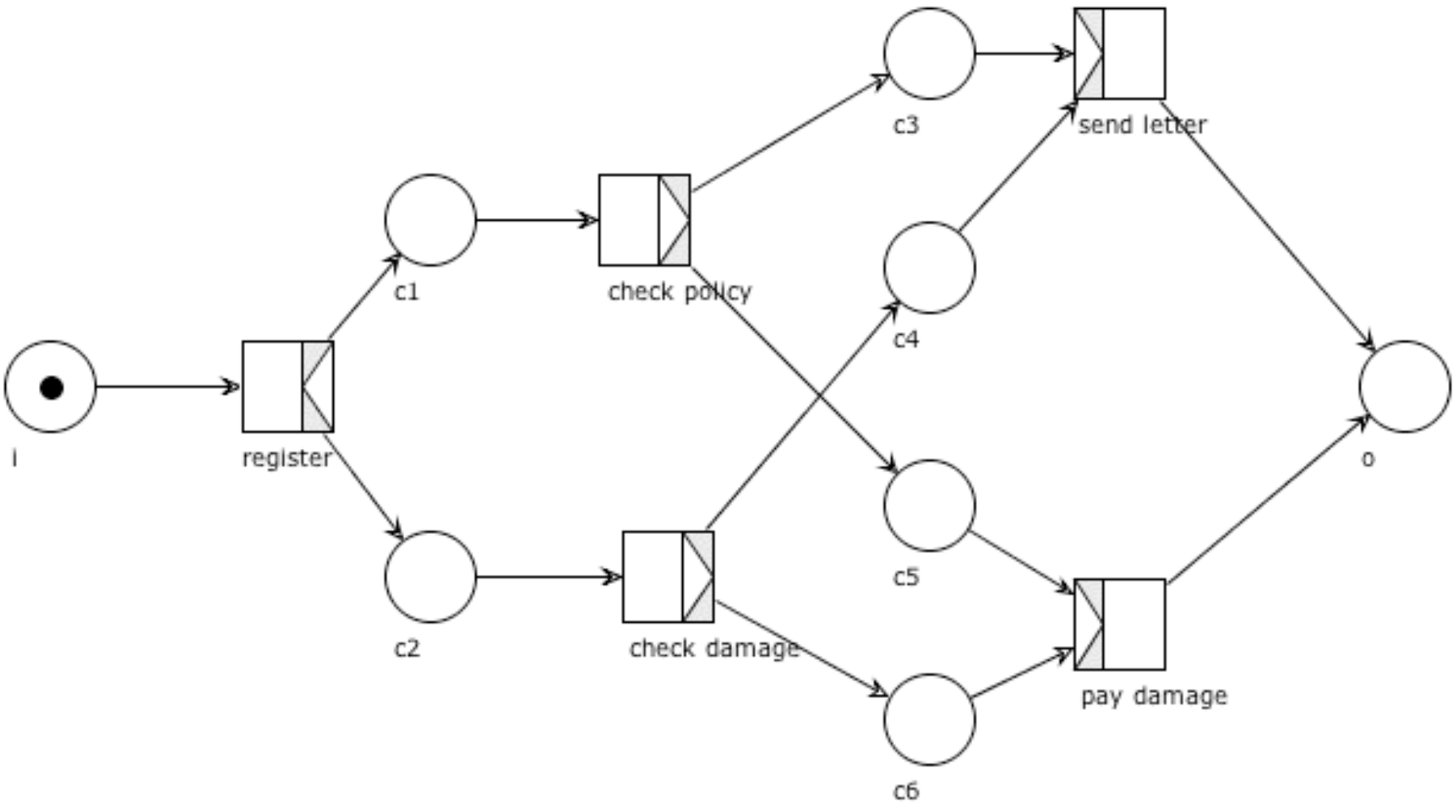
# Analyse the running example



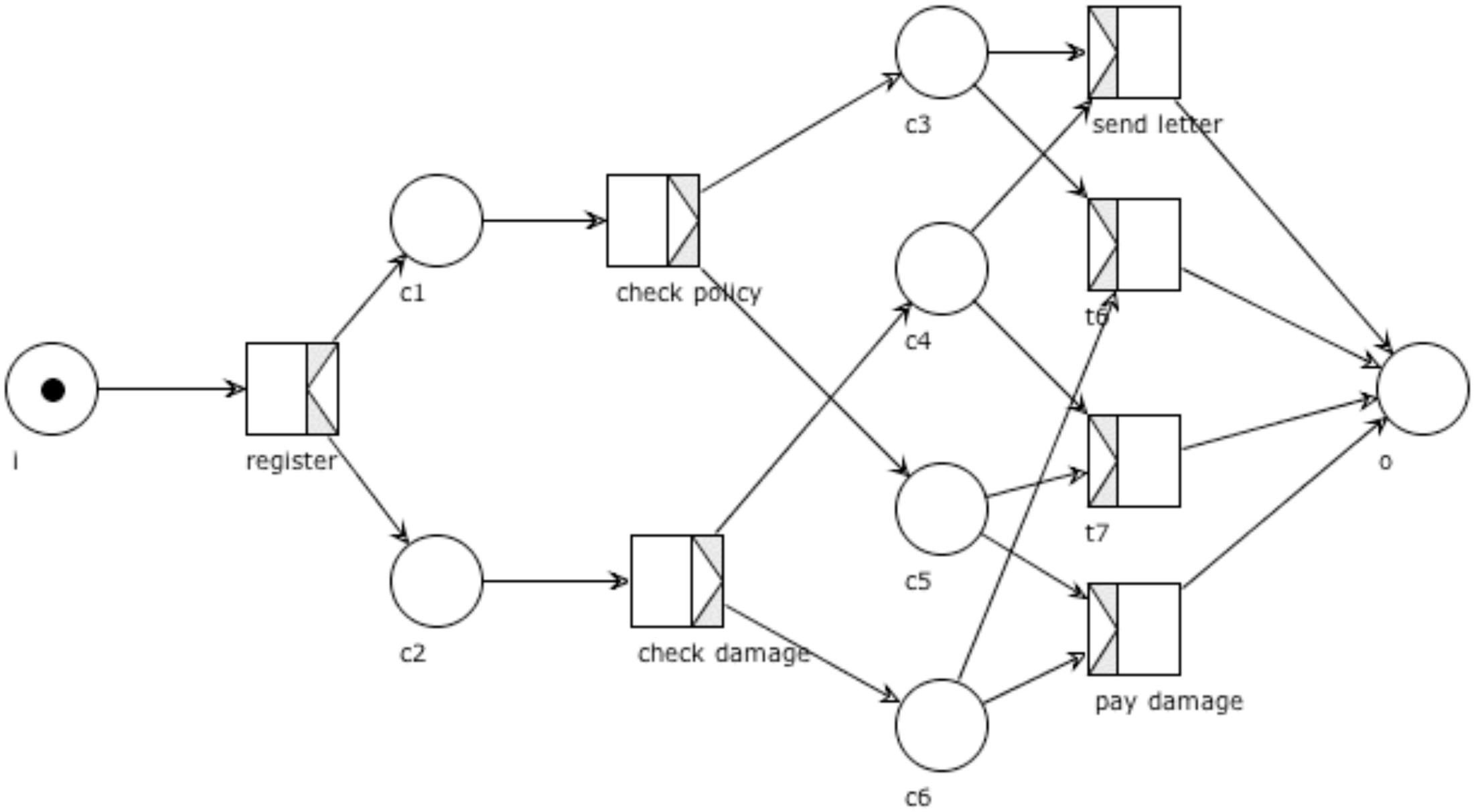
# Analyse the running example variant



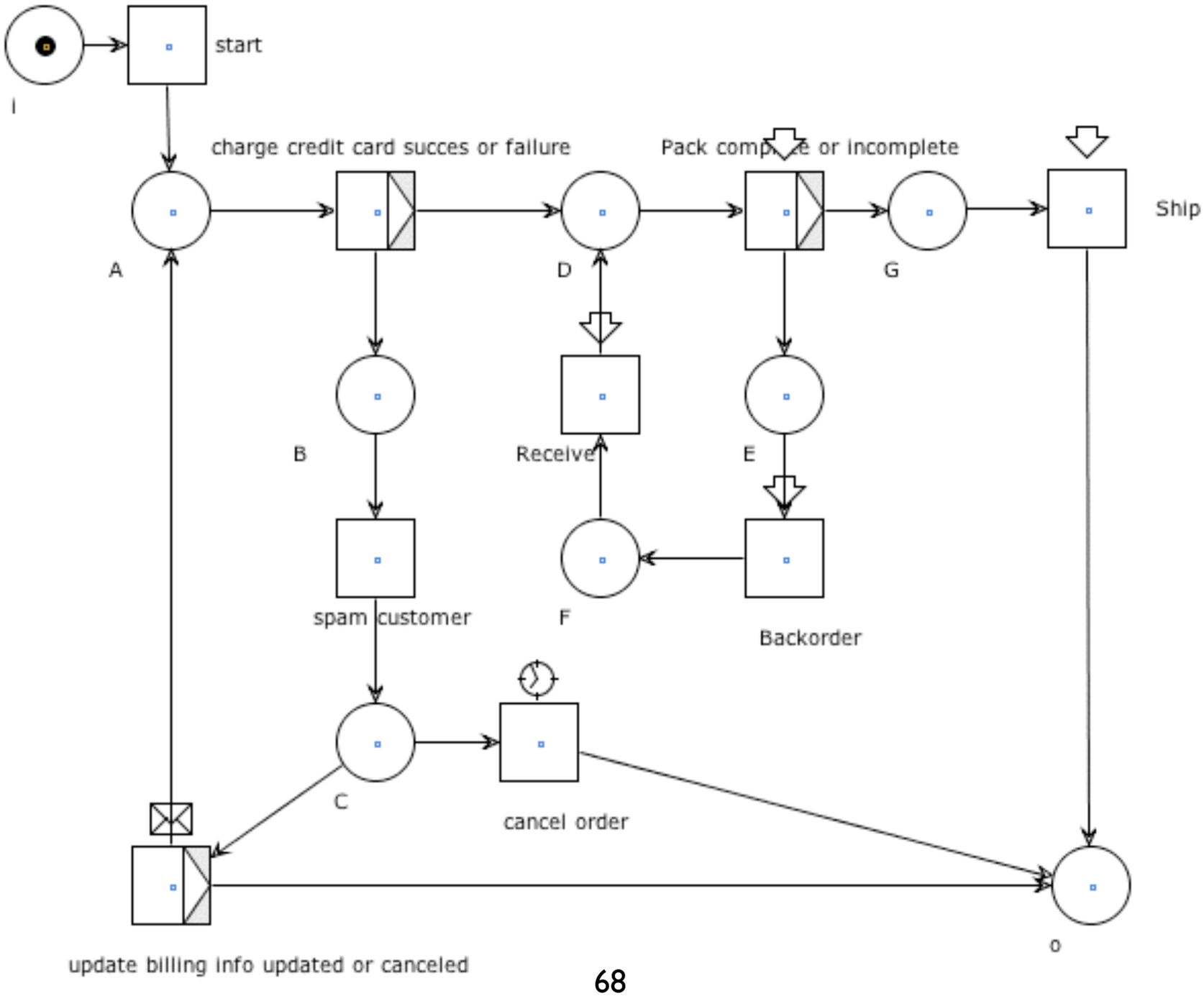
# Analyse this net



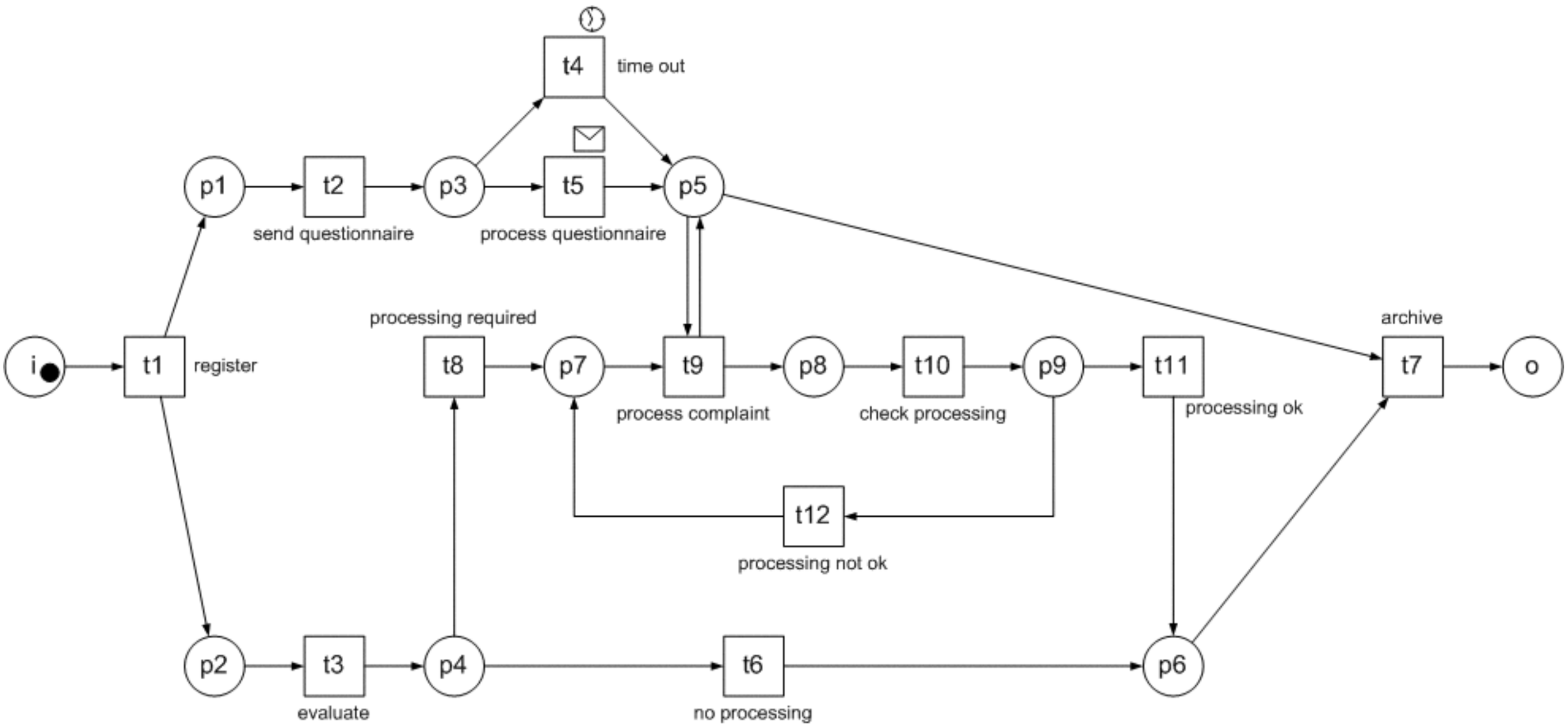
# Analyse this net



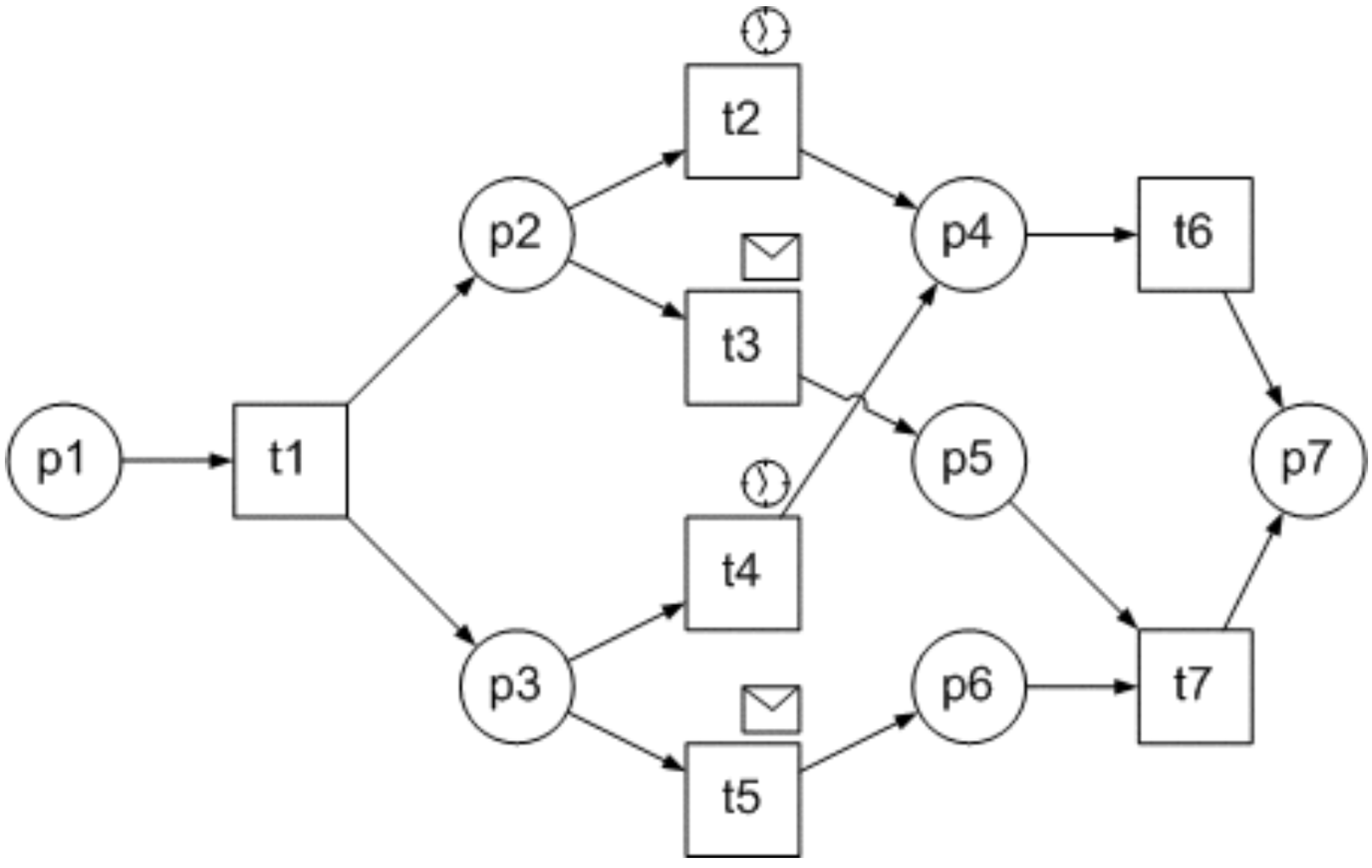
# Analyse this net



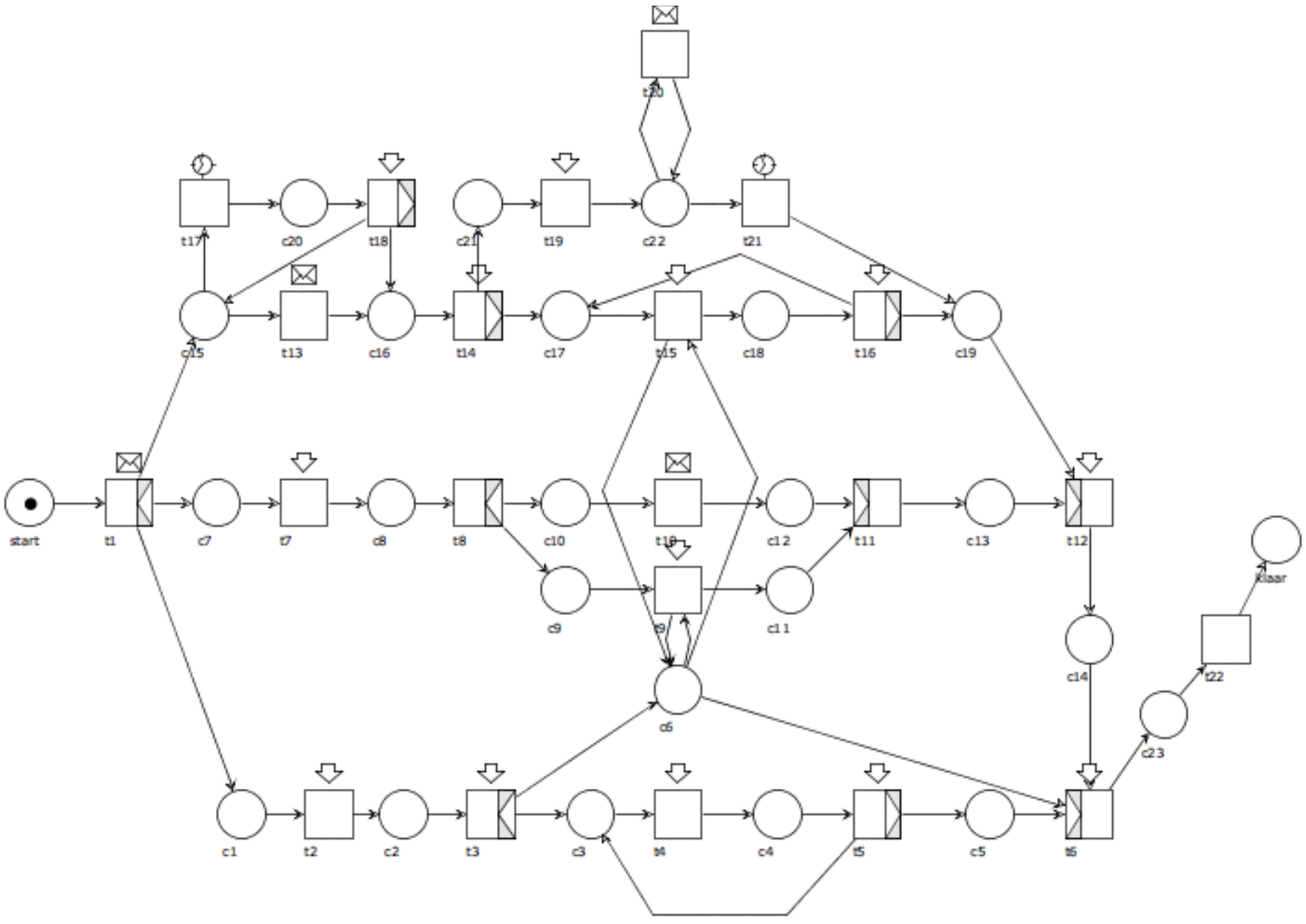
# Analyse this net



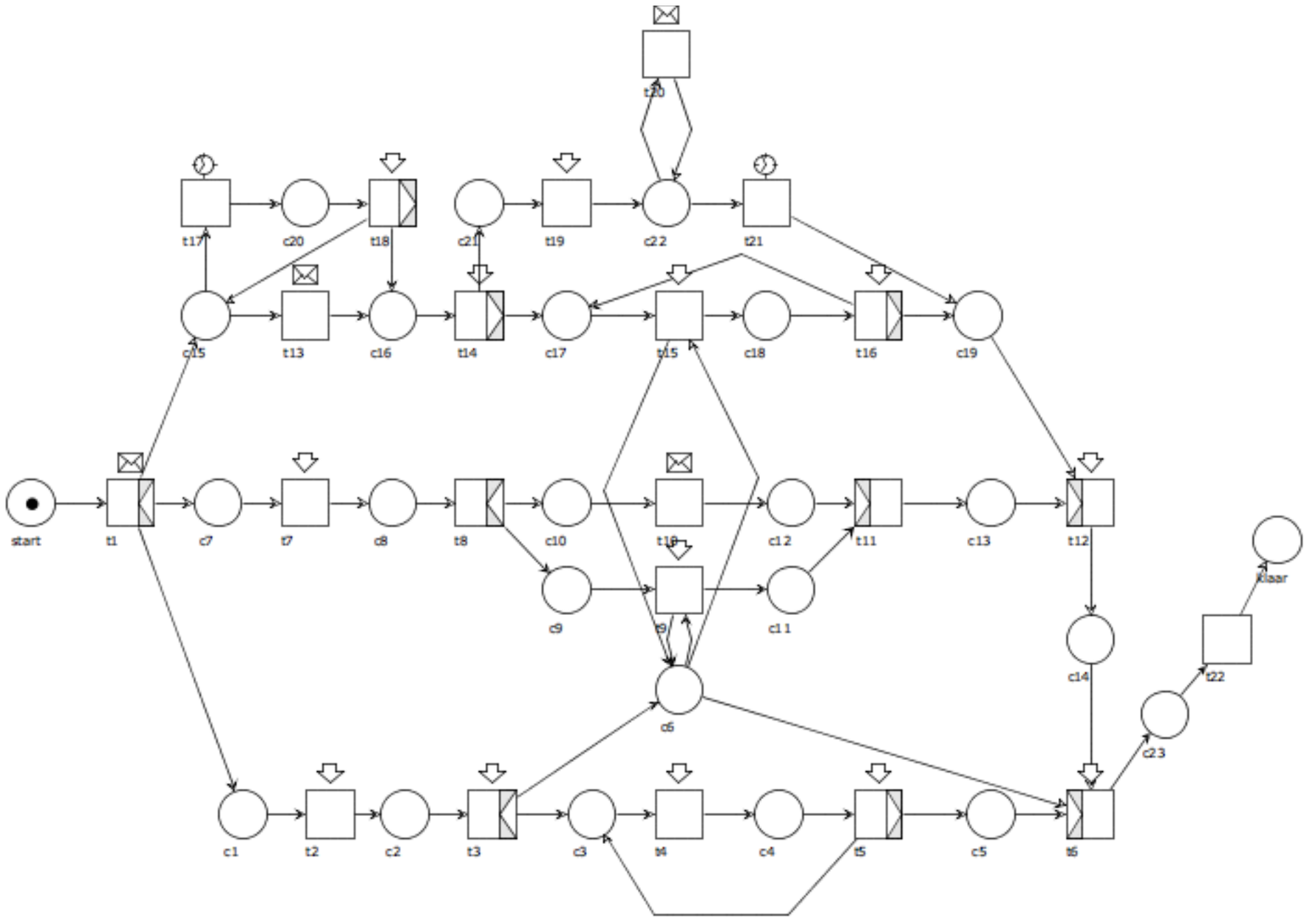
# Analyse this net



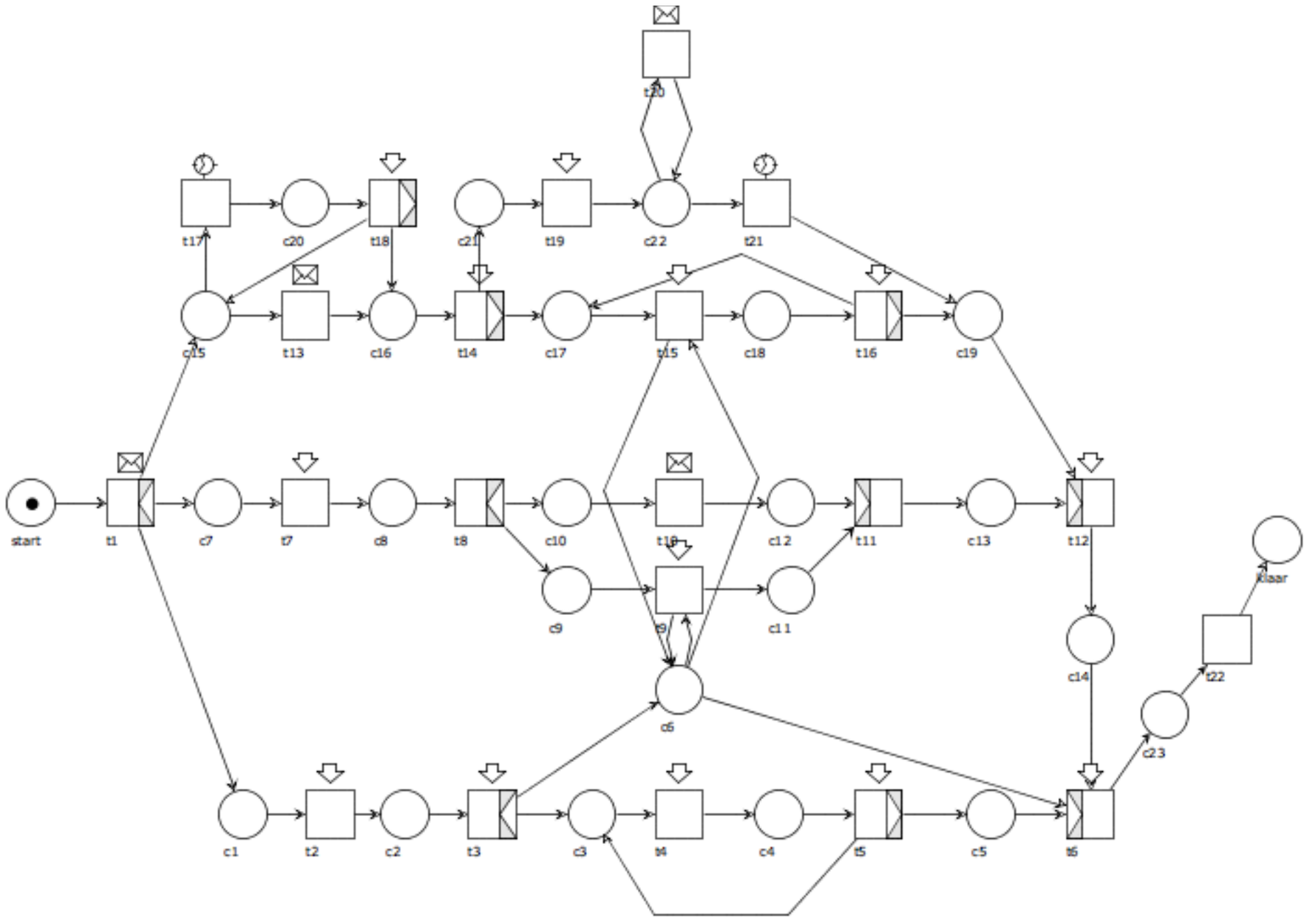
# Is this net free-choice?



# Is this net S-coverable?



# Is this net sound?



# Design and analysis of WF-nets

The workflow of a computer repair service (CRS) can be described as follows.
A customer brings in a defective computer and the CRS checks the defect and hands out a repair cost calculation back.
If the customer decides that the costs are acceptable, the process continues, otherwise she takes her computer home unrepaired.
The ongoing repair consists of two activities, which are executed, in an arbitrary order.
The first activity is to check and repair the hardware,
whereas the second activity checks and configures the software.
After each of these activities, the proper system functionality is tested.
If an error is detected another arbitrary repair activity is executed,
otherwise the repair is finished.

Model the described workflow as a sound workflow net.

# Design and analysis of WF-nets

A hospital wants to establish a rating workflow for their doctors. To make the workflow reliable two different roles are assigned.
The first one is a referee from the newly created quality assurance department while the second one represents the managing director of the hospital.
Both roles execute all of their tasks independently from each other.

The referee starts a new case regarding a certain doctor by interviewing patients.
Since a patient interview workflow is already established, it is simply integrated in the new workflow.
Meanwhile, the director asks an external expert to review the work of the doctor under rating.
Unfortunately, since the expert only gets a low expenses fee, it can happen that the expert is not responding in time.
If that happens, another expert has to be asked (who could also not respond in time, i.e. the procedure repeats).
If an expert finally sends an expertise, it is received by the director and forwarded to the referee.
The referee files the results containing the patient interviews as well as the expertise and afterward creates a report.
While the referee is doing this, the manager fills a cheque to pay the expenses of the expert.

Model the described workflow as a sound workflow net.