Relating Reversible Petri Nets and Reversible Event Structures

Categorically

Hernan C. Melgratti, Claudio Antares Mezzina, and G. Michele Pinna

NiRvAna annual meeting @Ca' Foscari

Background 1/3

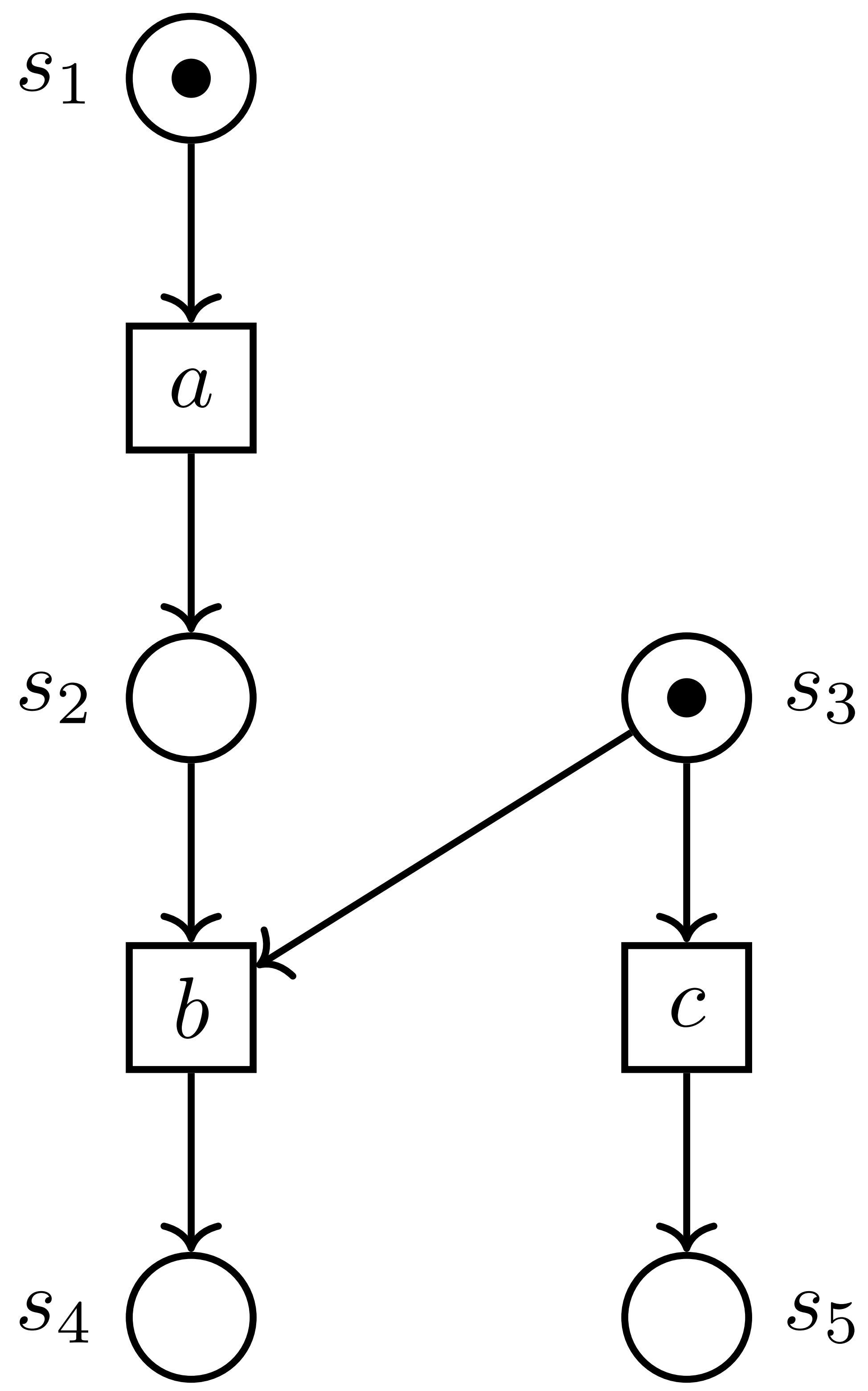
Two well-known models to describe concurrent systems:

- Event structures
    - Event occurrences and constraints on events
    - Denotational view of a system
- Petri nets
    - Consumption / production of data from repositories
    - Places, tokens, transitions
    - Operational view of a system

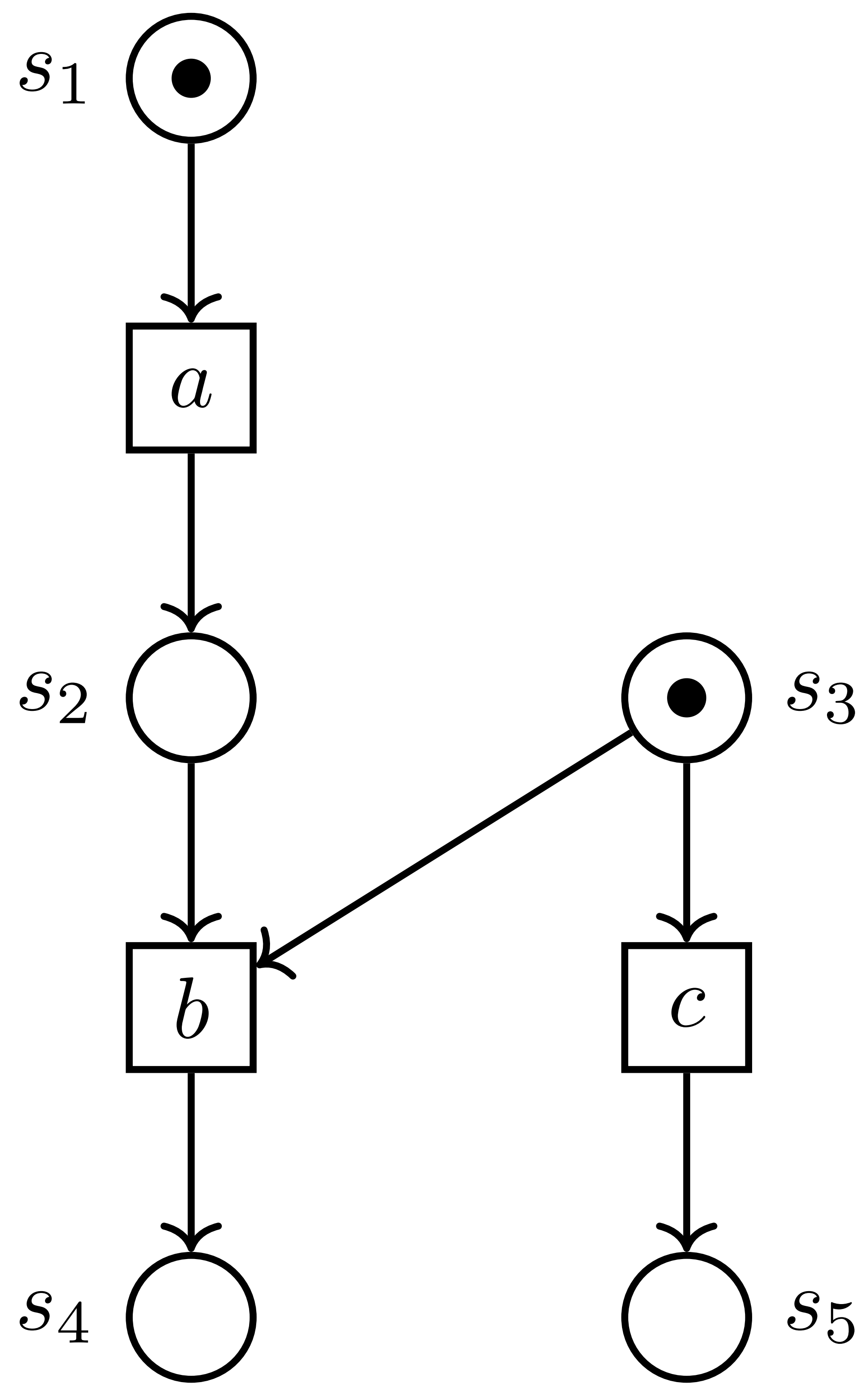
Background 2/3

- A seminal work of Winskel showed a relation between Occurrence Nets (ON) and Prime Event Structure (PES)
- A PES describes describes a computational process as
  - a set of events whose occurrence is constrained by two relations
  - causality $<$ and
  - (symmetric) # conflicts.

Example

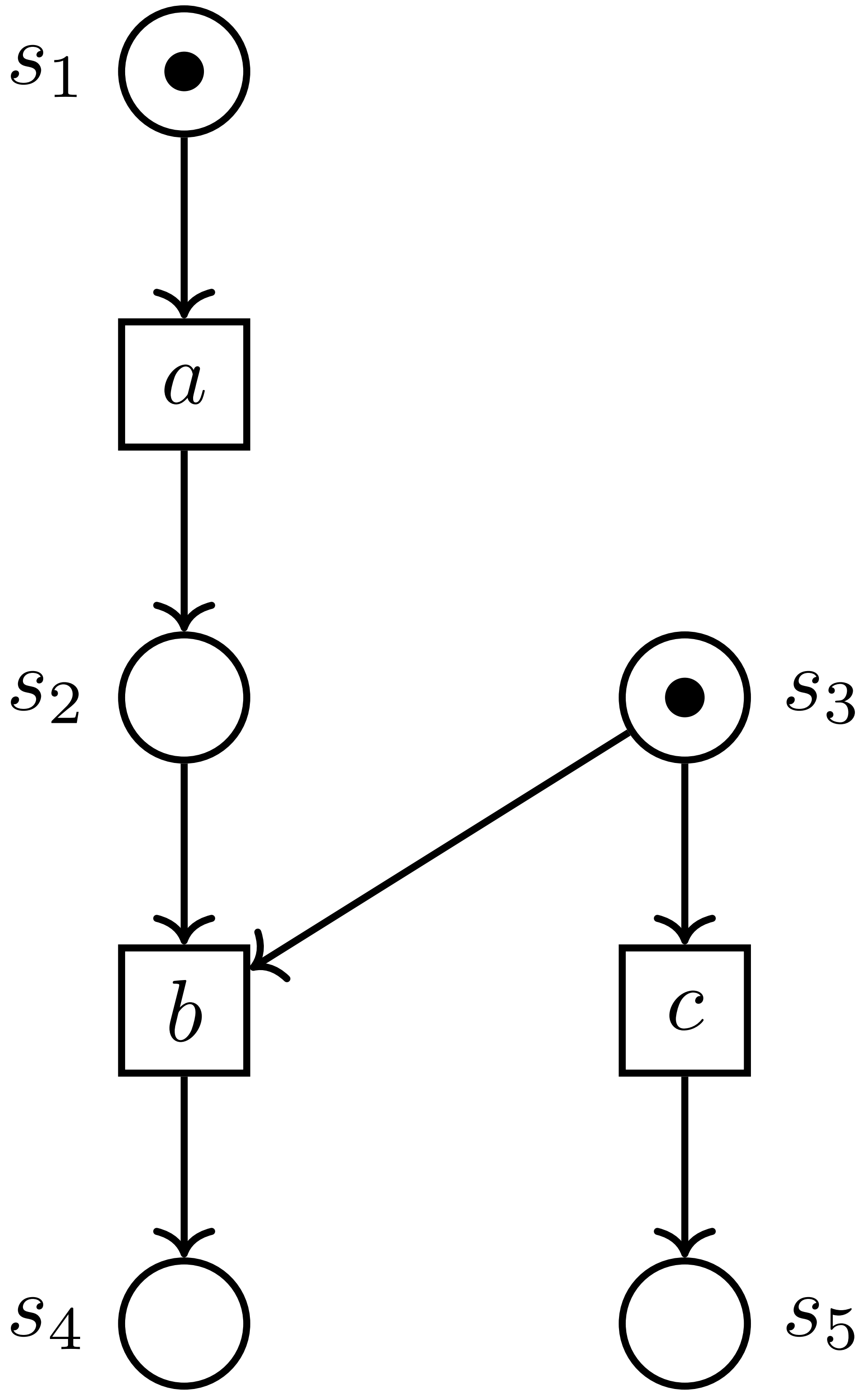


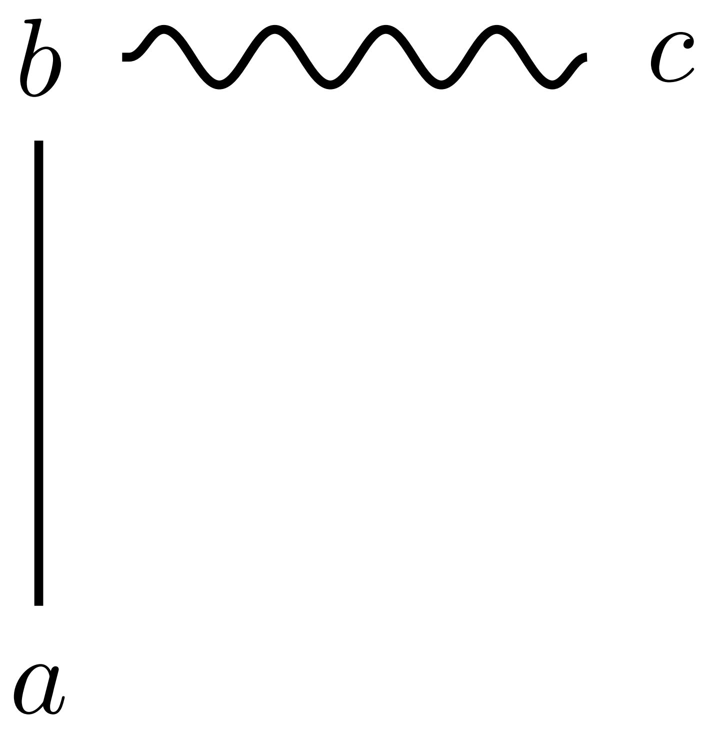
Example



a < b   b#c

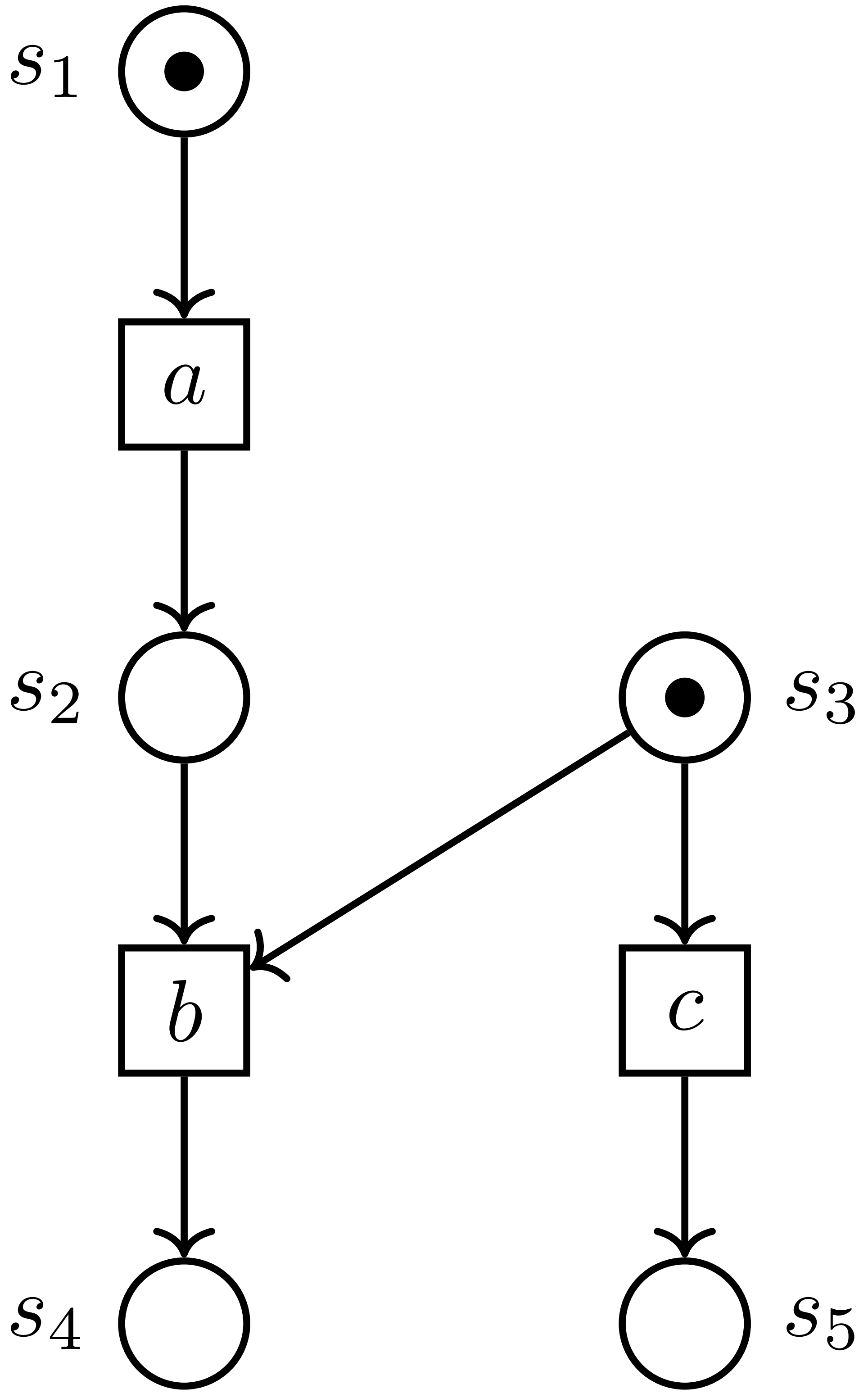
Example





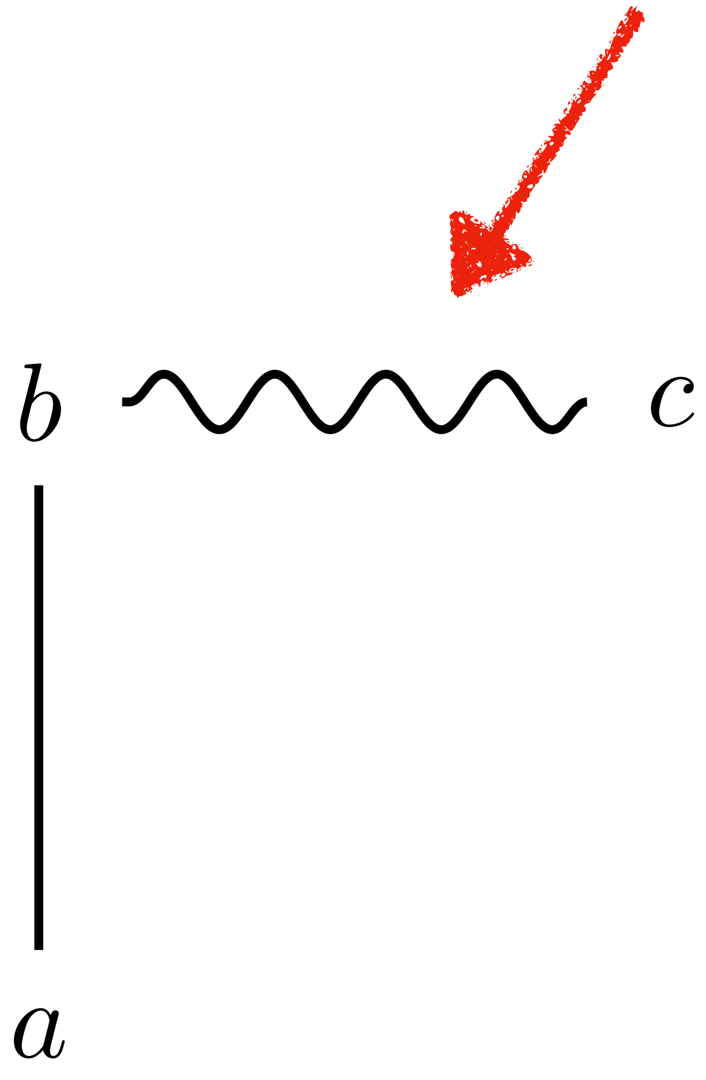
a < b   b#c

Example

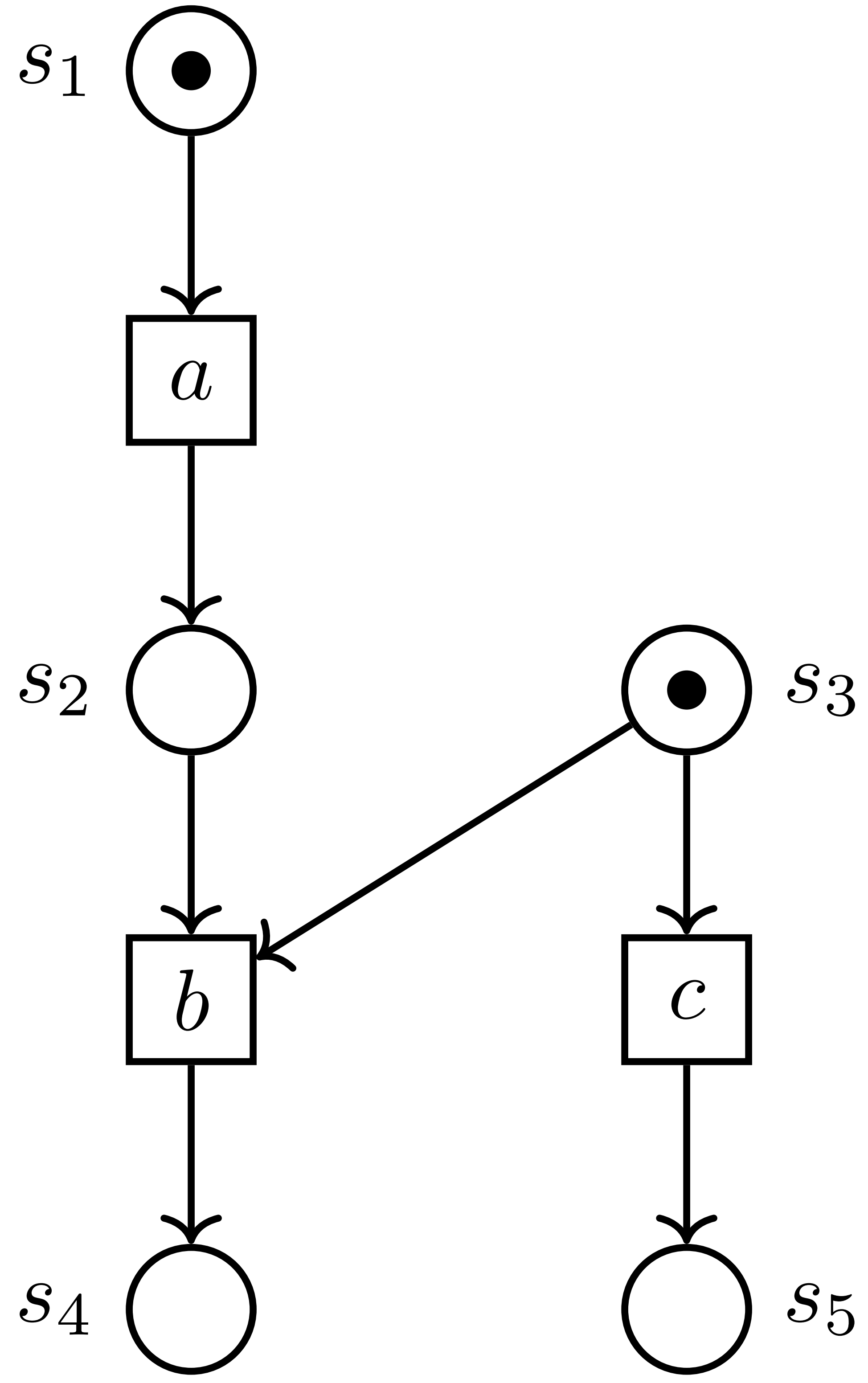


a < b   b#c

b and c are in conflict

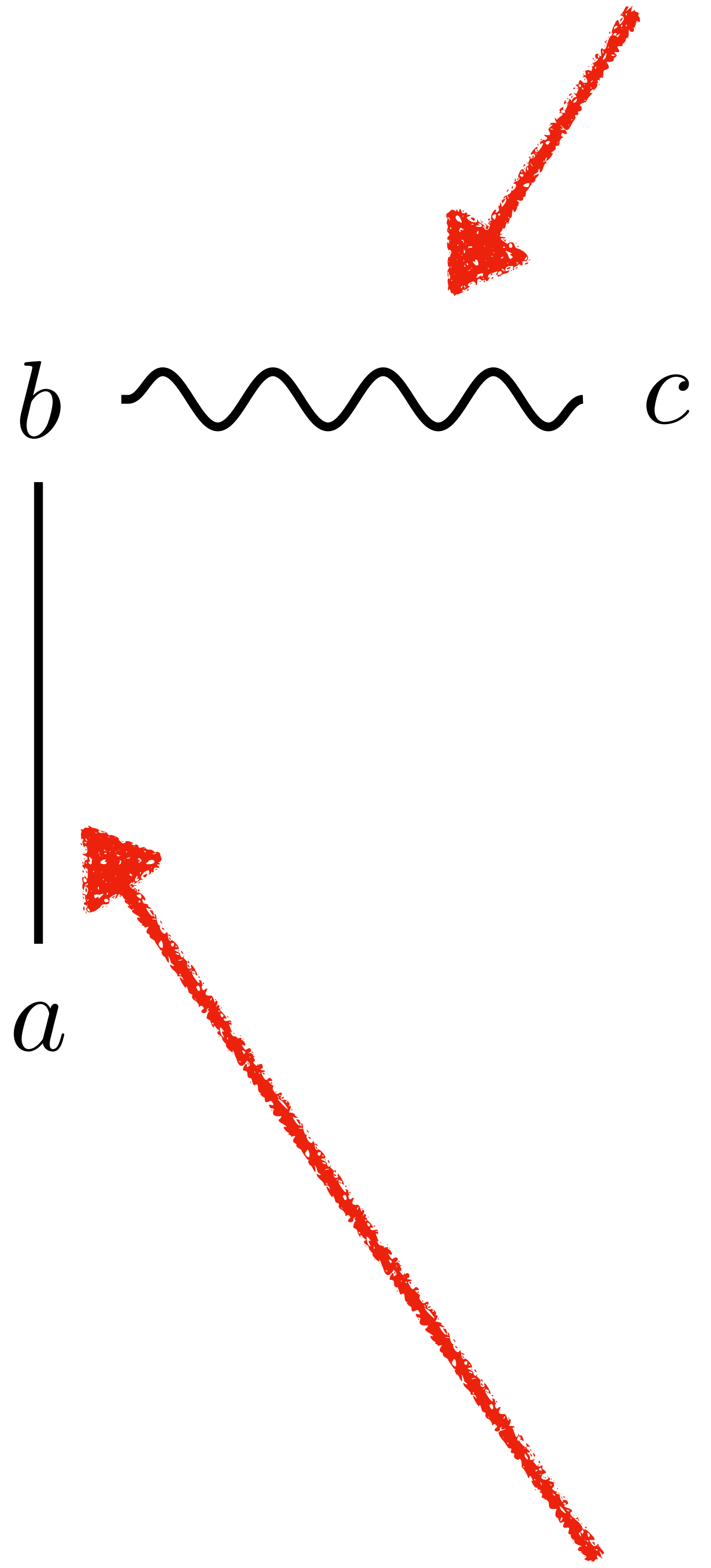


Example



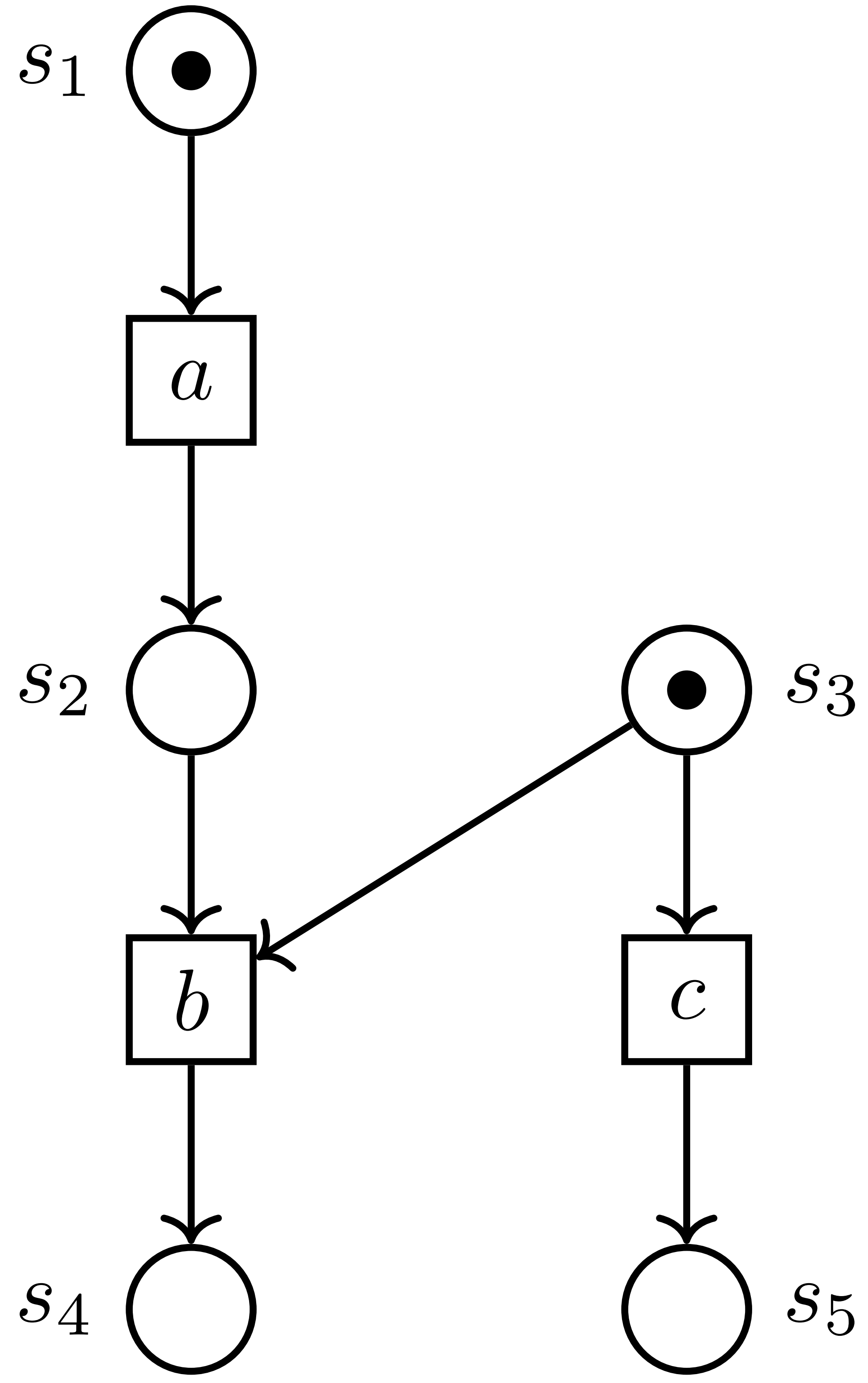
a < b   b#c

b and c are in conflict



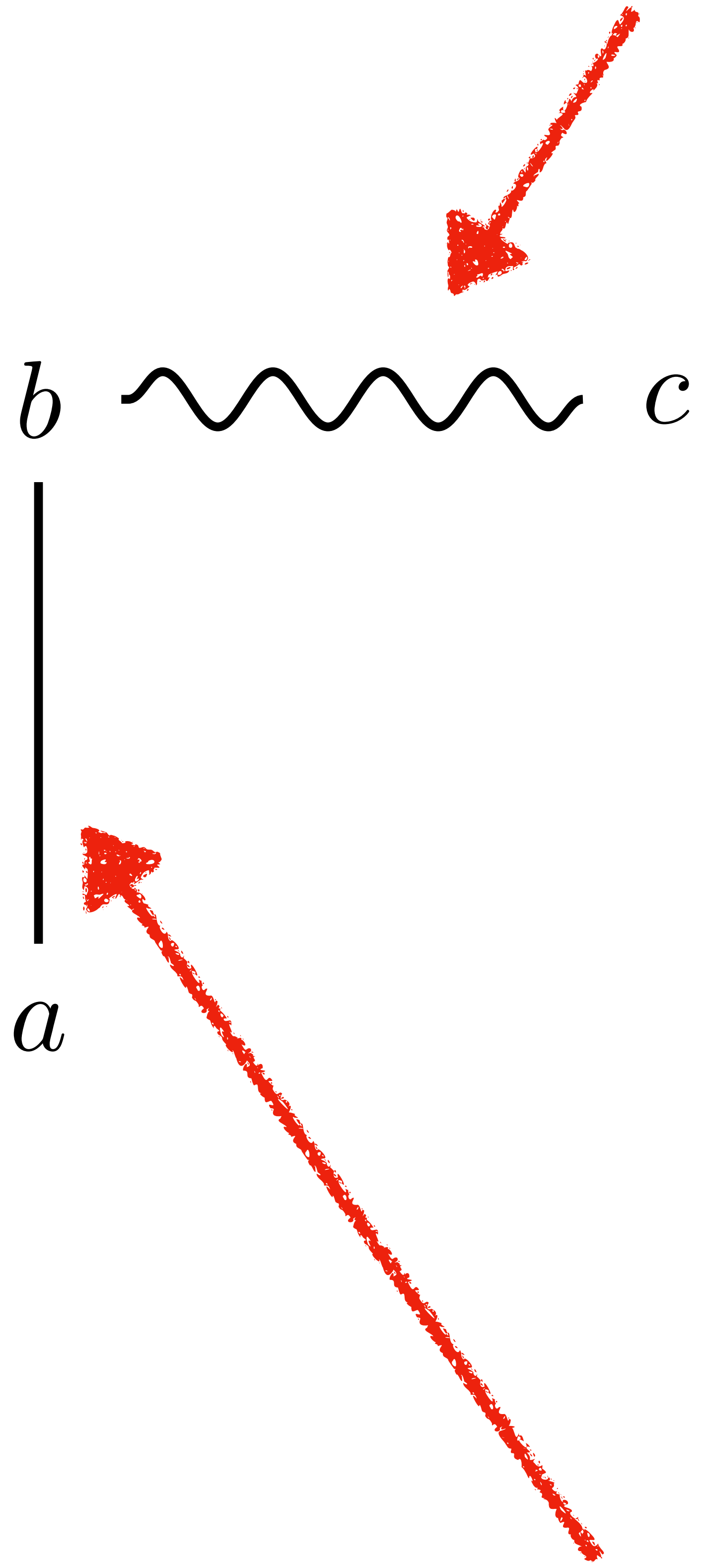
b causally depends on a

Example

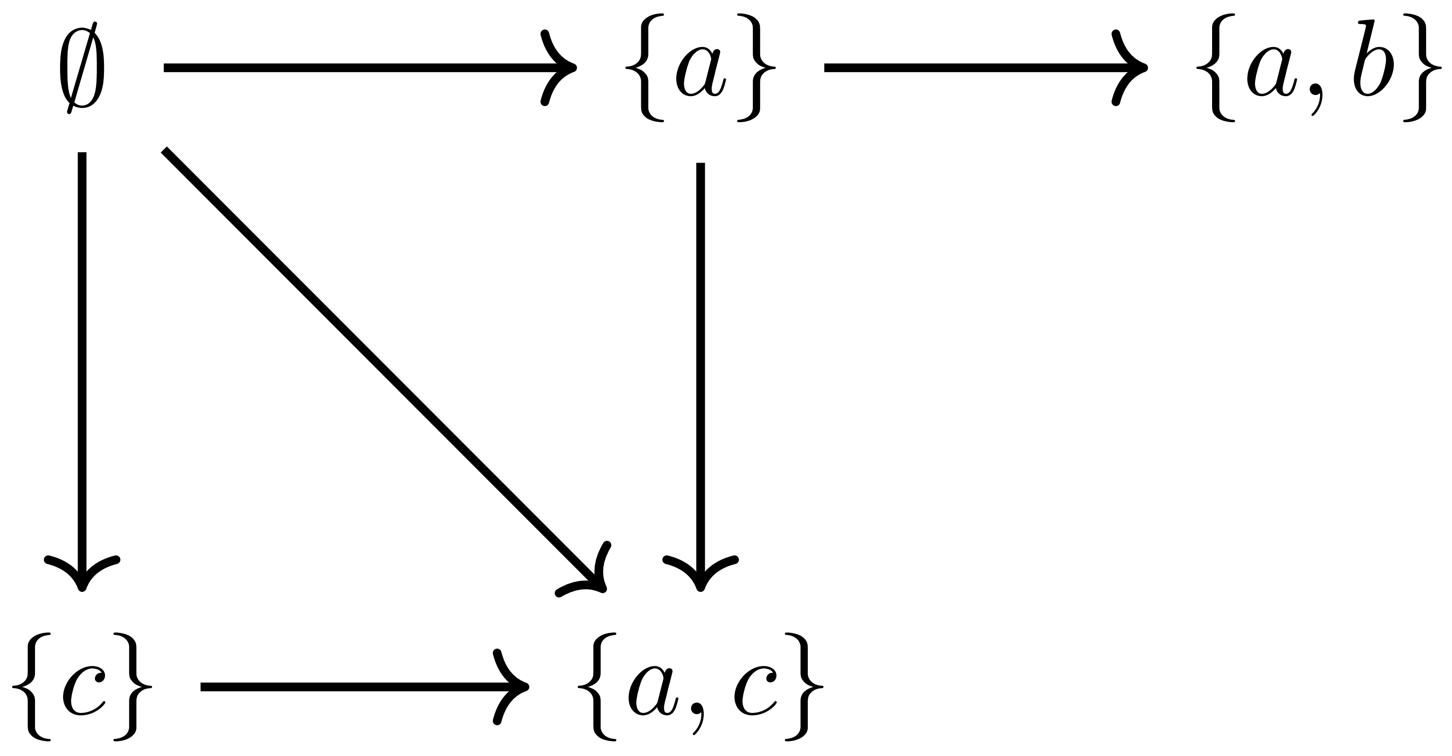


a < b   b#c

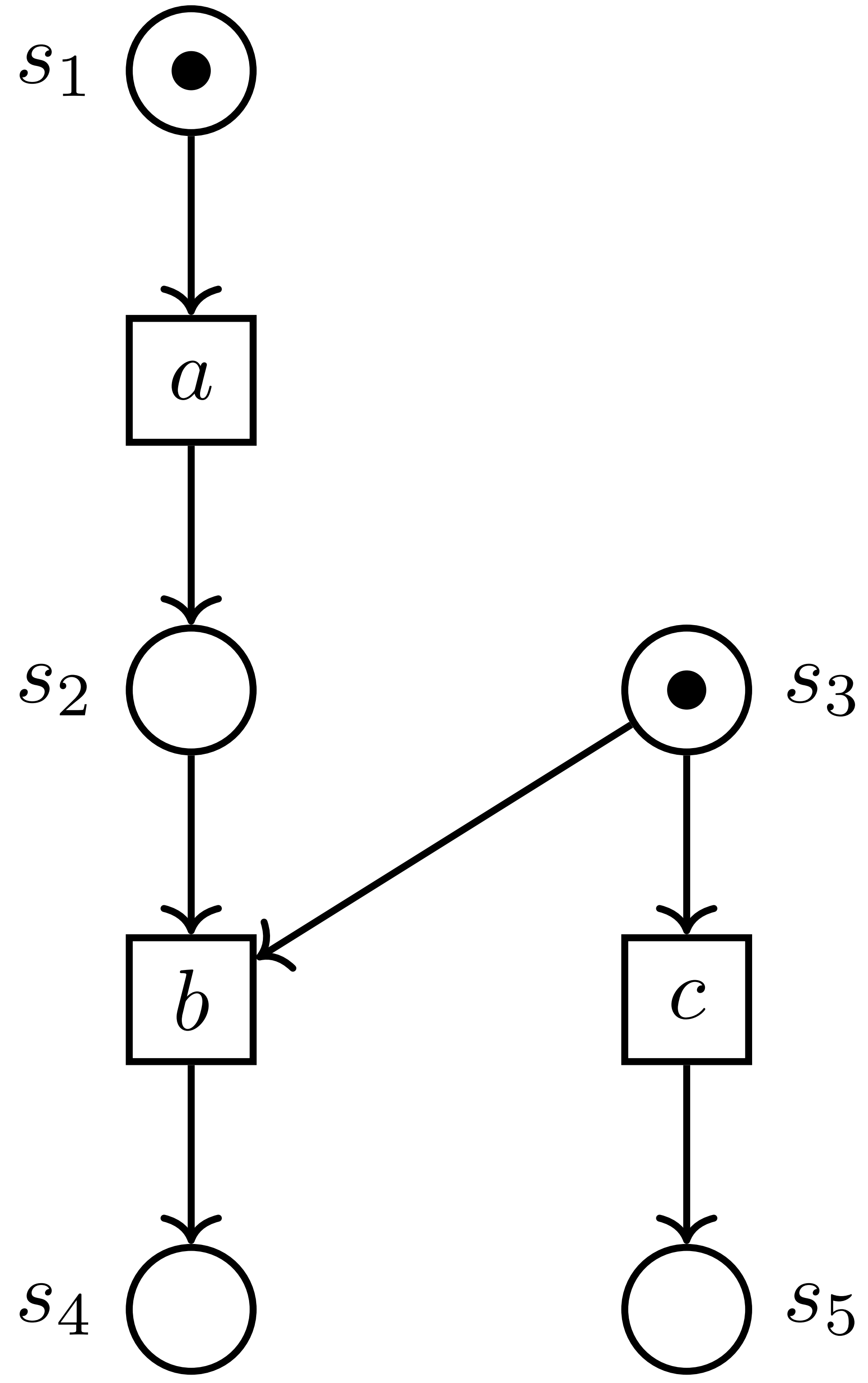
b and c are in conflict



b causally depends on a

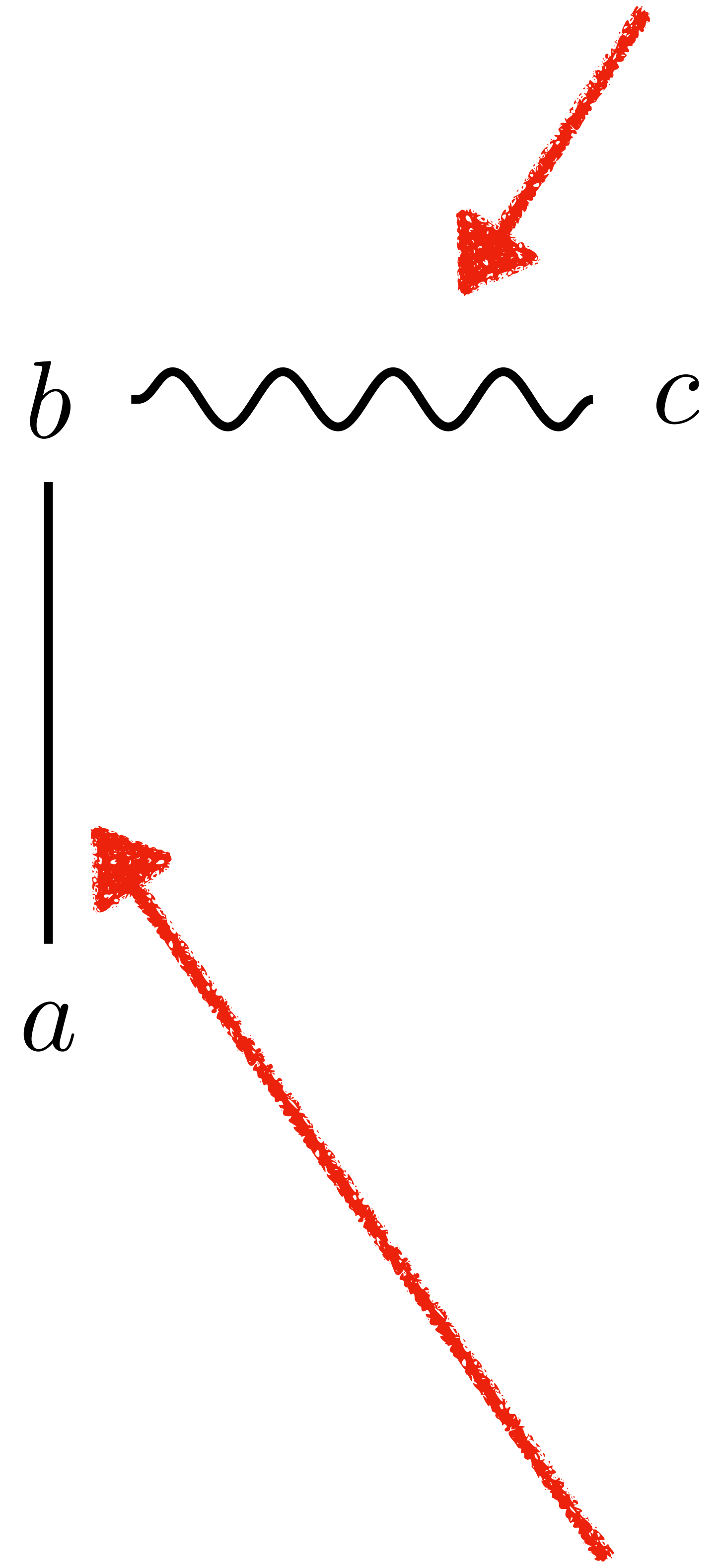


Example

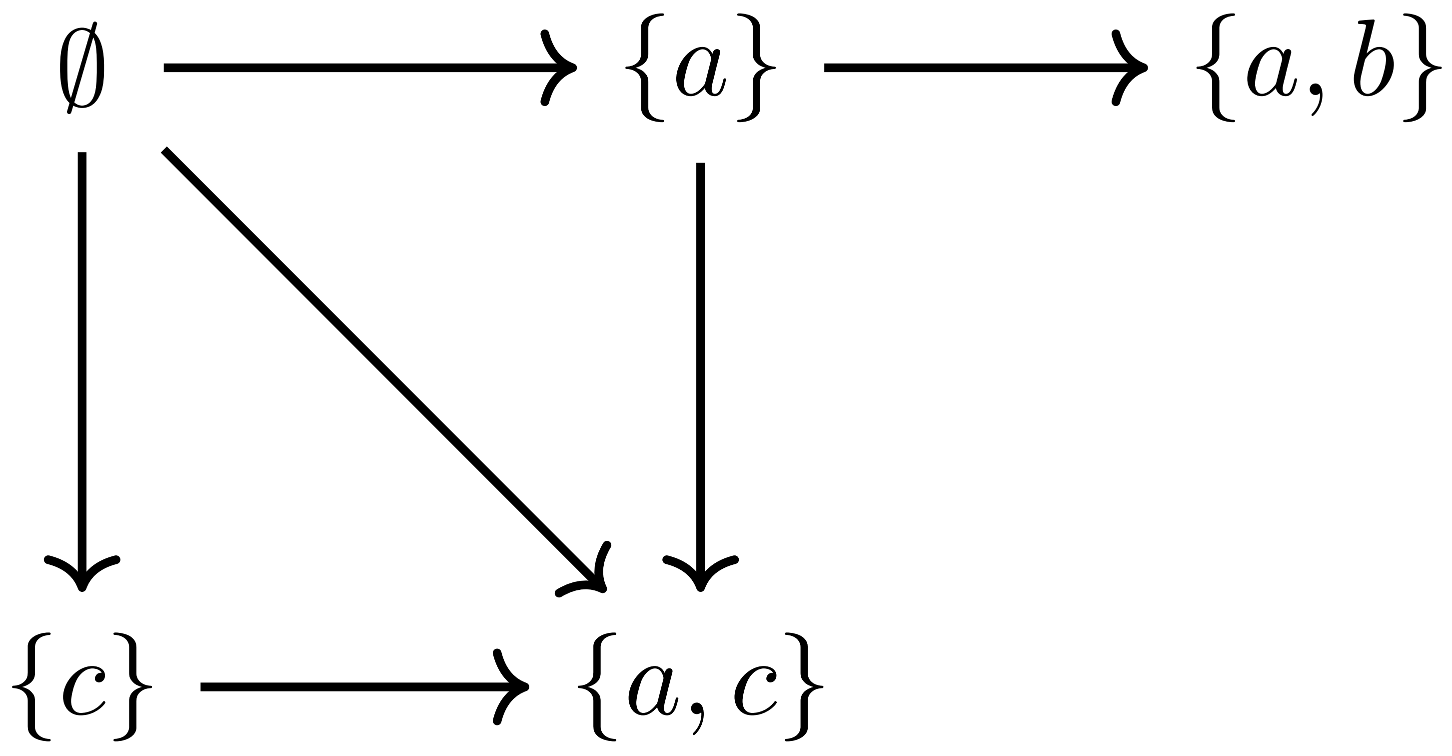


a < b   b#c

b and c are in conflict



b causally depends on a



Since b and c are in conflict there is no configuration containing both

If b is present in a configuration then also a is present

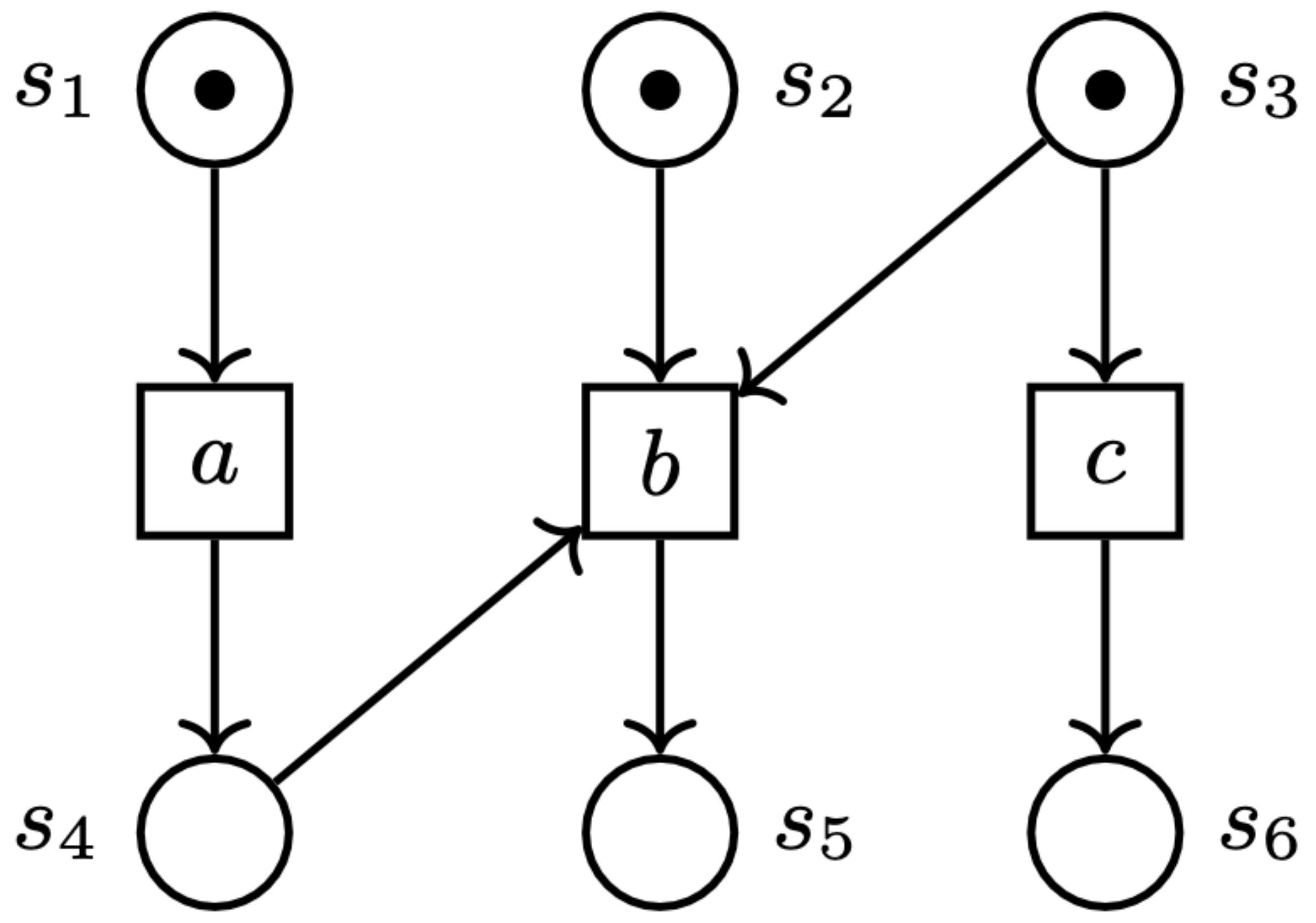
Background 3/3

- PESs have been extended to account for reversible computing
  - accomodate the undoing of executed actions by removing events from configurations
  - accounts for different kinds of reversibility: backtracking, causal-respecting (transactions / checkpoint rollback) and out-of-order (biochemical reactions)
  - Reversible PESs (rPESs) add two more relations to PESs
    - reverse causation ($\prec$) and
    - prevention ($\triangleright$)
- A recent work shows that the operational model of (reversible) PES can be recovered by reversible Causal Nets (runs),

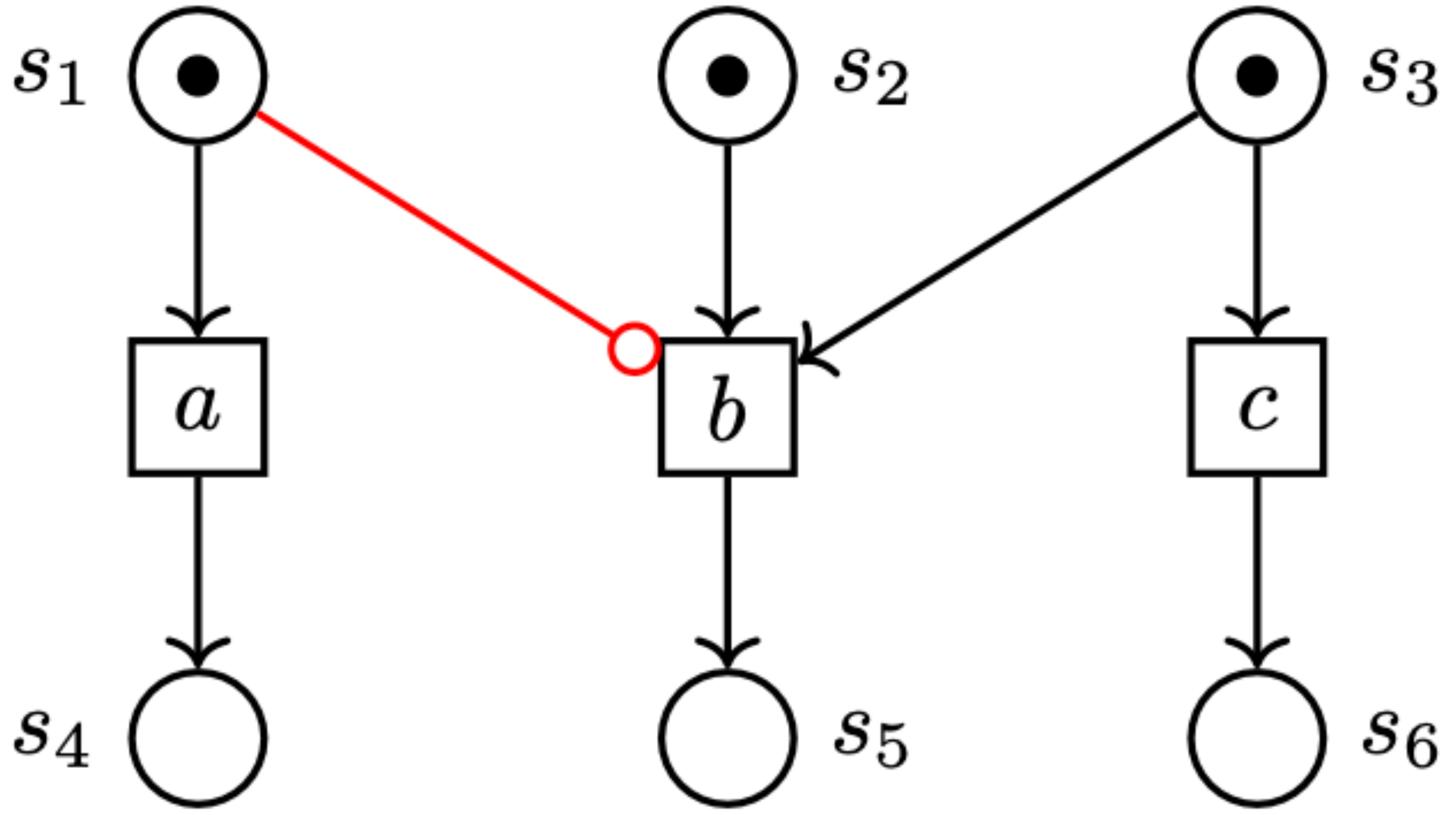
Causal Nets

- Causal nets are occurrence nets where causality is expressed via inhibitor arcs a not derived by the usual flow relation
- Occurrence nets are Petri nets in which
    - the net seen as a graph has no cycles;
    - every place (circle) has at most one incoming transition (e.g., no backward conflict)
    - no node is in self-conflict
- Every PT net can be unfolded into an occurrence net (Winskel81)

Causal Nets



$a < b$ and $b \# c$

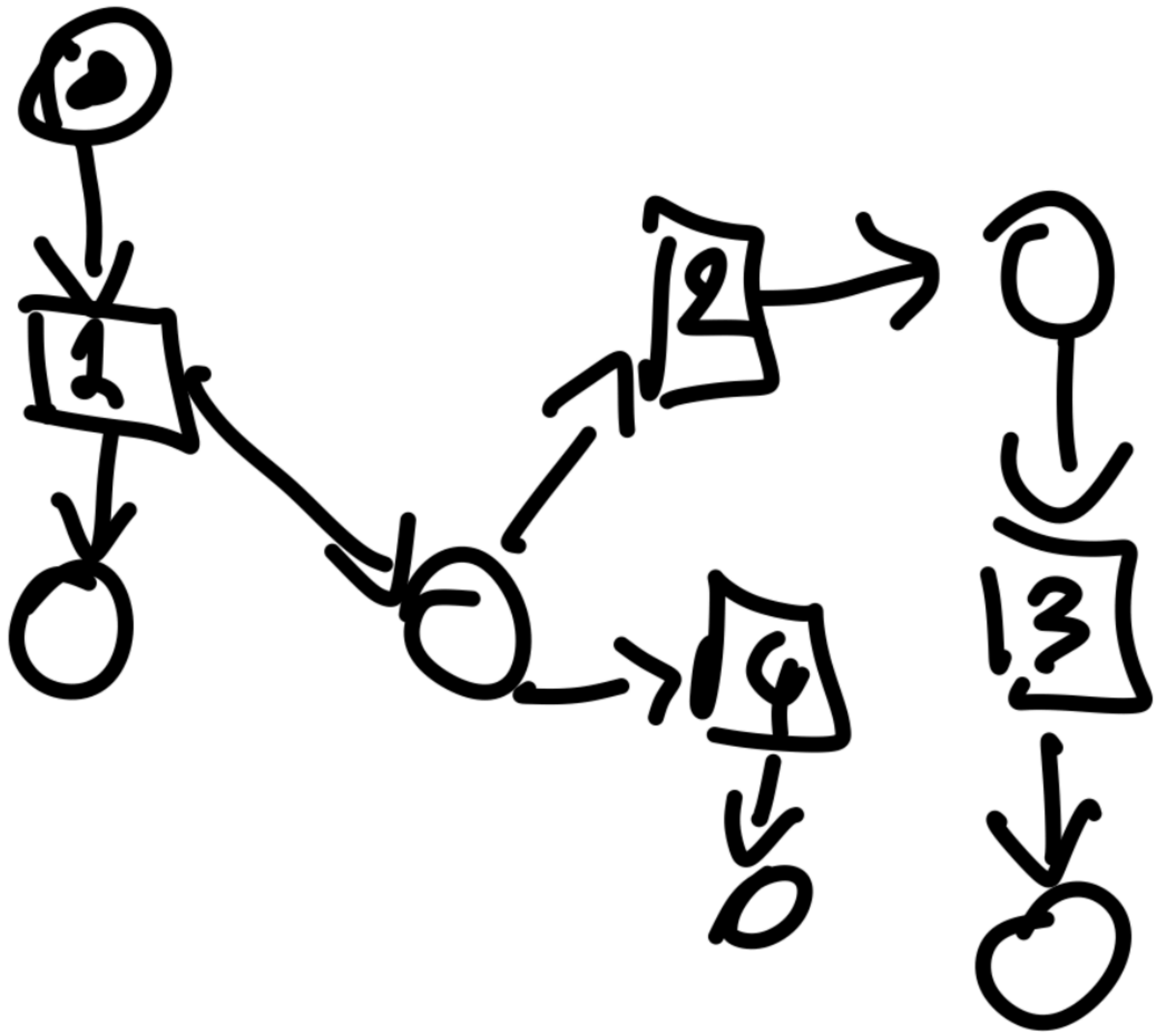


$a < b \quad b \# c$

In causal nets causality is recovered from inhibitor arcs instead of the usual overlap between post and presets of transitions (e.g., flow relation)

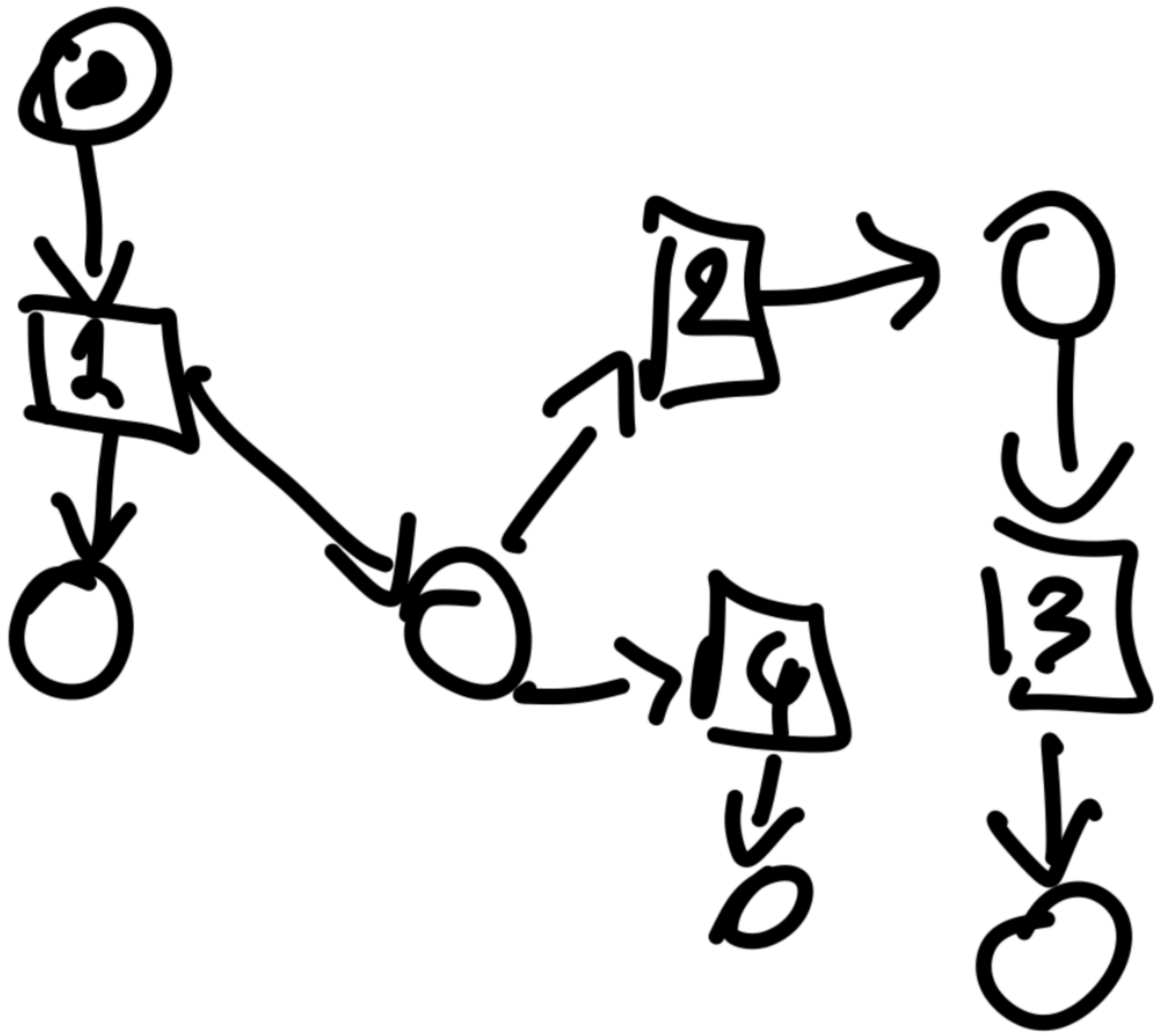
Inhibitor arcs prevents the firing of a transition if a token is present in some place of the net

Causal Nets - example

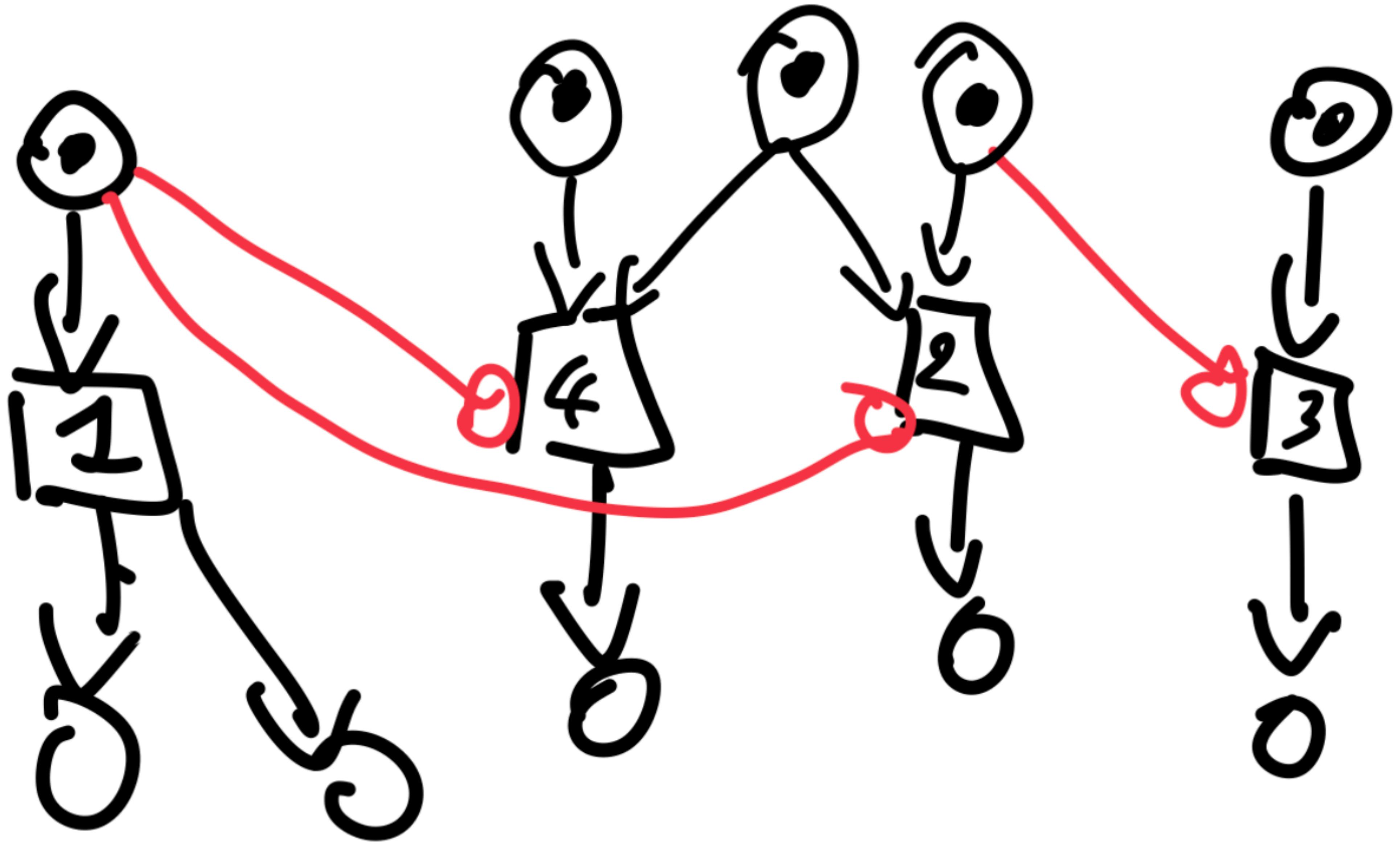


1 < 2
1 < 4
2 < 3
2#4

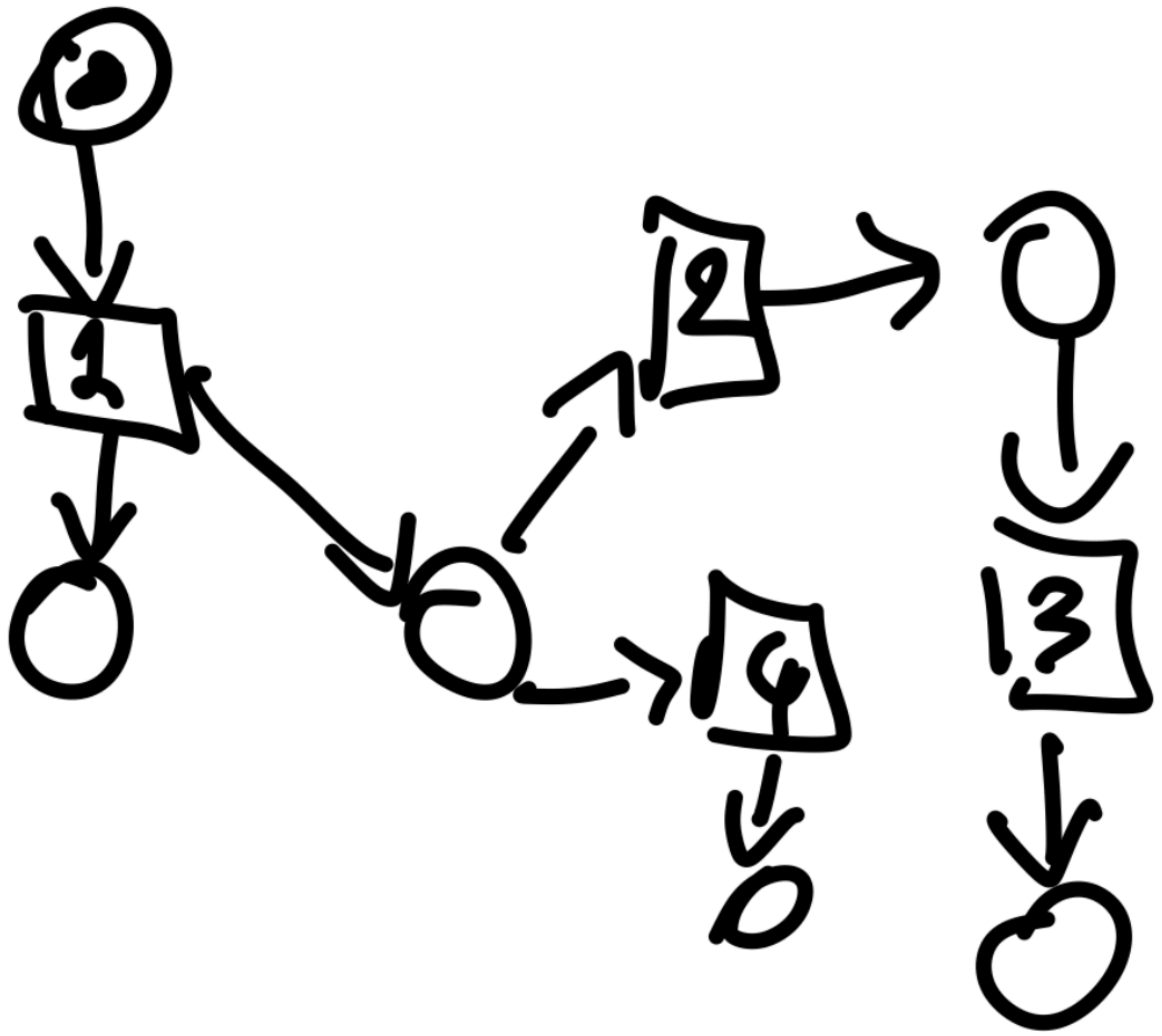
Causal Nets - example



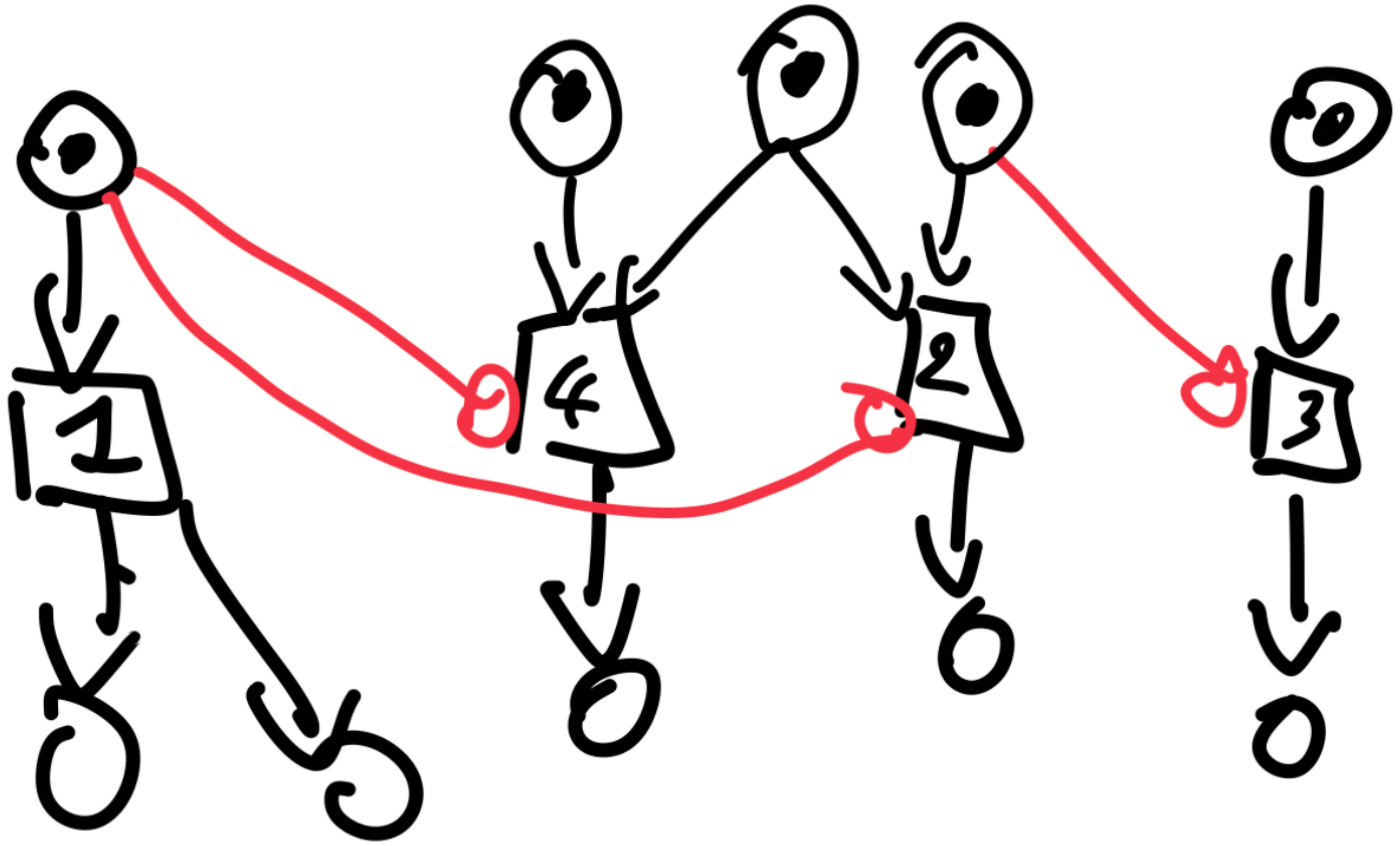
1 < 2
1 < 4
2 < 3
2#4



Causal Nets - example

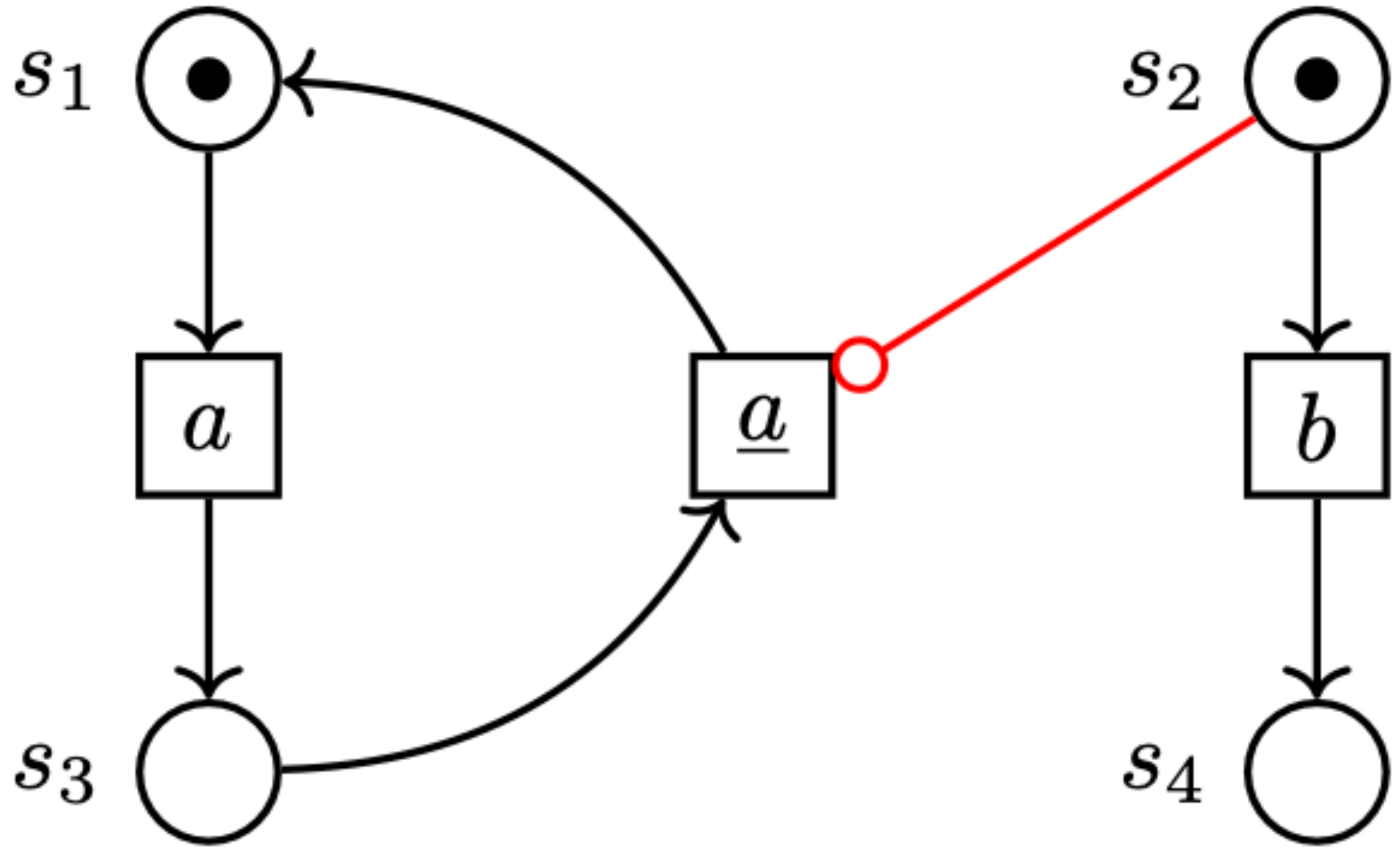


1 < 2
1 < 4
2 < 3
2#4

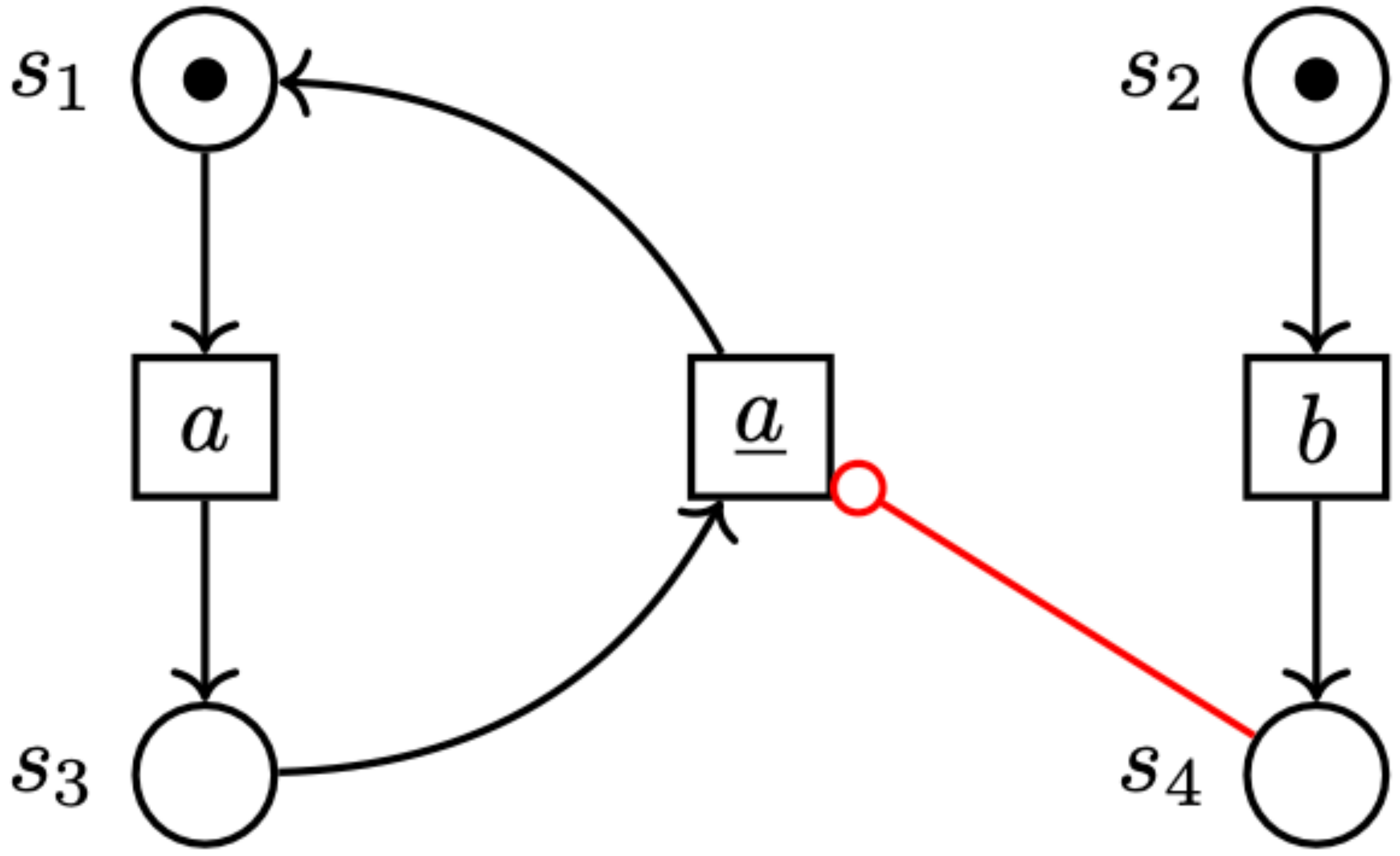


1 < 2
1 < 4
2 < 3
2#4

Reverse causal nets



$b \prec \underline{a}$



$b \triangleright \underline{a}$

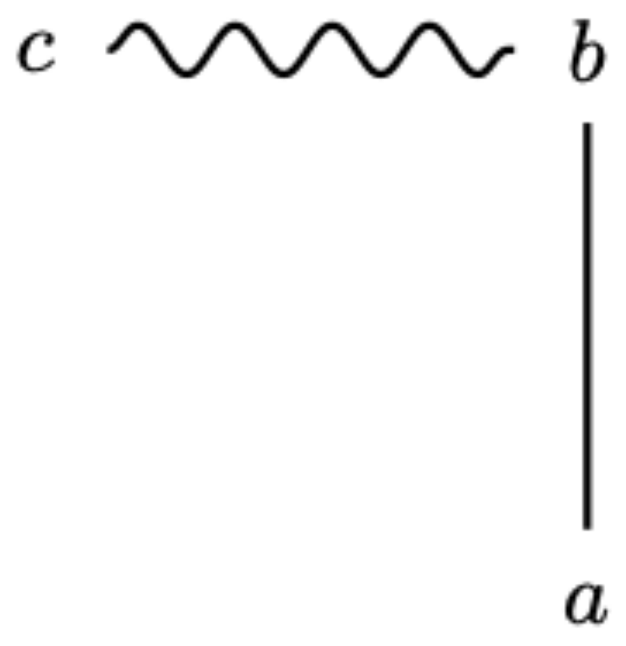
Inhibitor arcs can be also used to model

- Reverse causality (a cannot reversed until b occurs)
- Prevention (a can be undone if b has not happened)

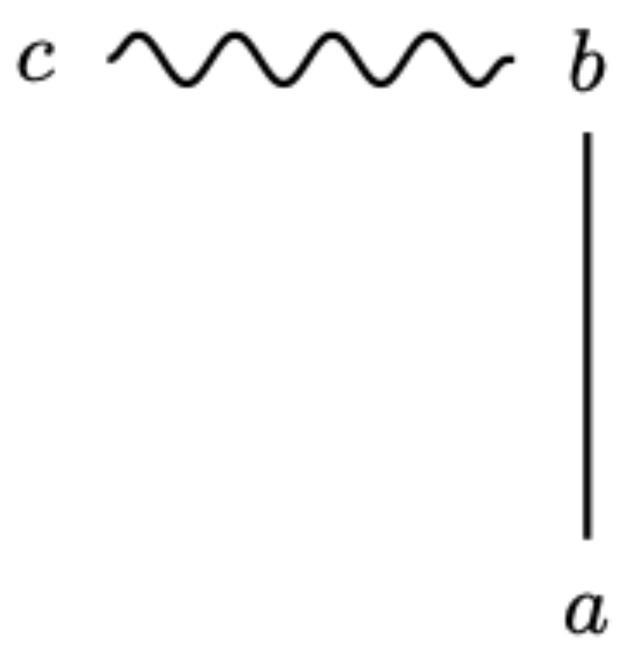
What about asynchrony?

- Asymmetric ESs relax the notion of conflict by considering *weak causality*
- Intuitively, an event e weakly causes the event e' (written $e \nearrow e'$) if e' can happen after e but e cannot happen after e'
- This can be considered as an asymmetric conflict because e' forbids e to take place, but not the other way round
- Symmetric conflicts can be recovered by making a pair of conflicting events to weakly cause each other
    - e # e' == ($e \nearrow e'$) and ($e' \nearrow e$)

AESs

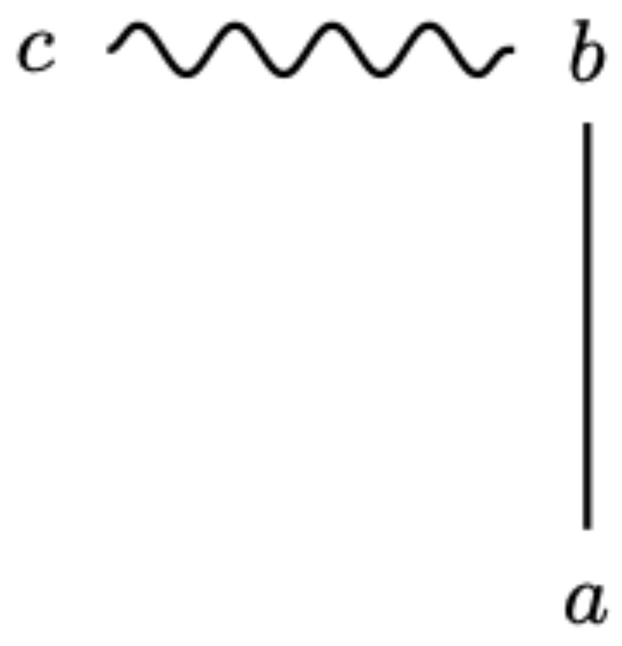


AESs

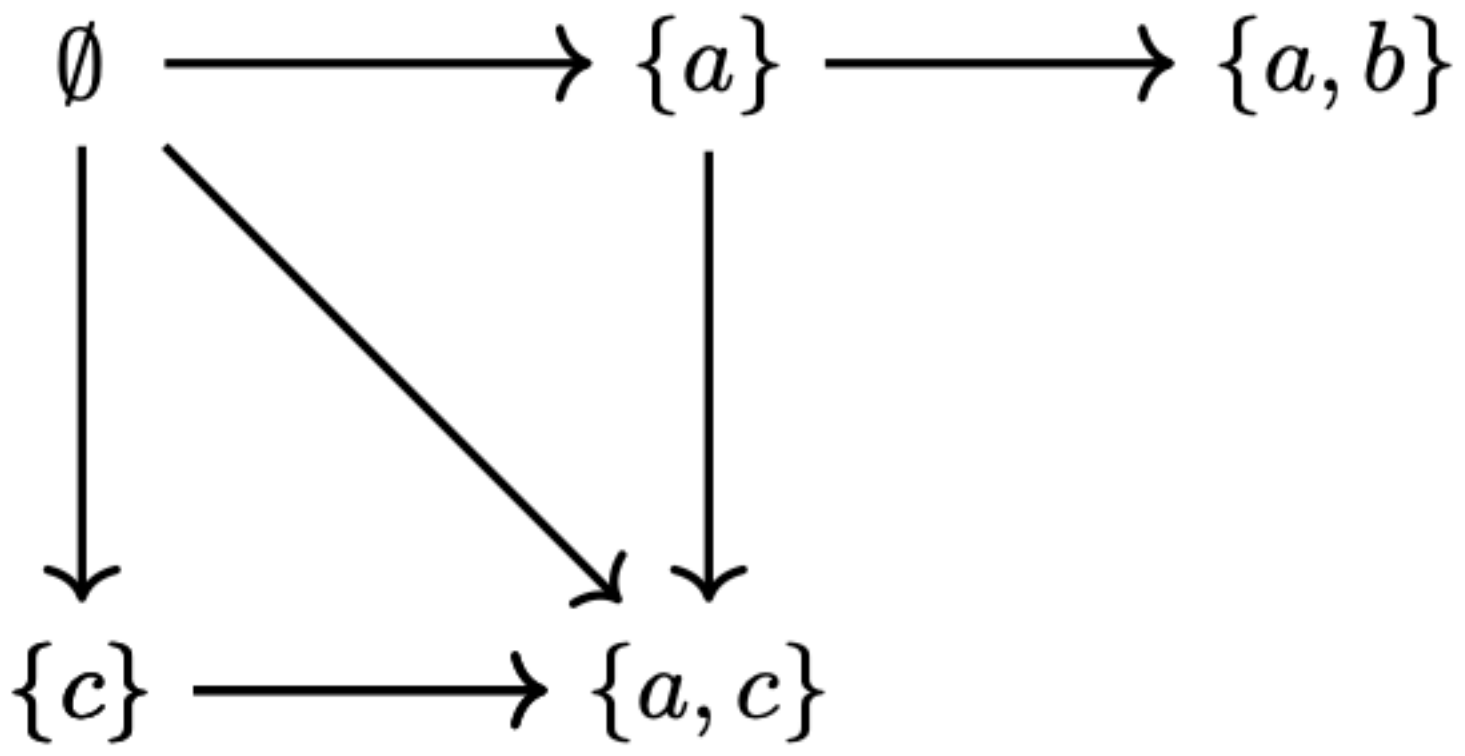


a < b   b#c

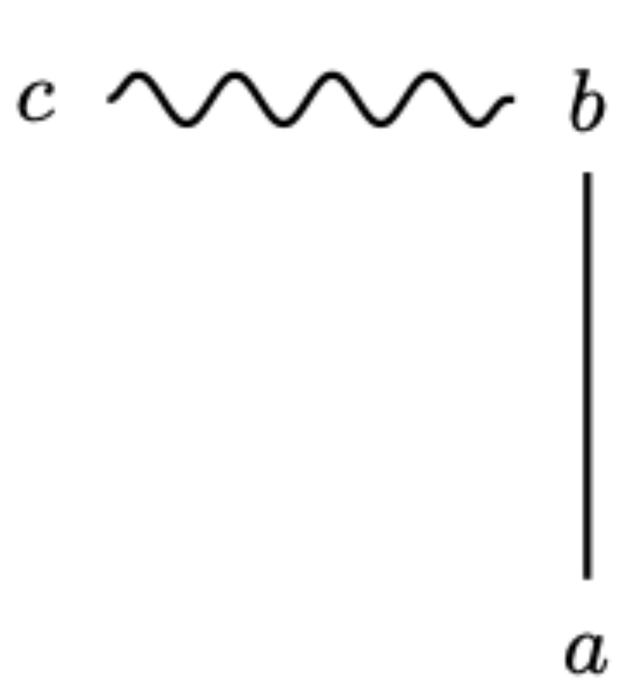
AESs

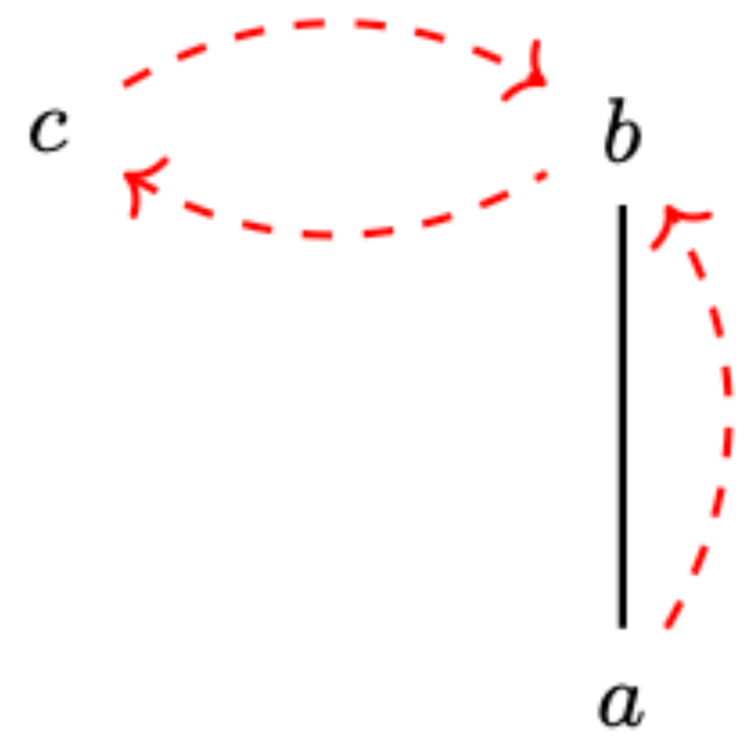


a < b   b#c

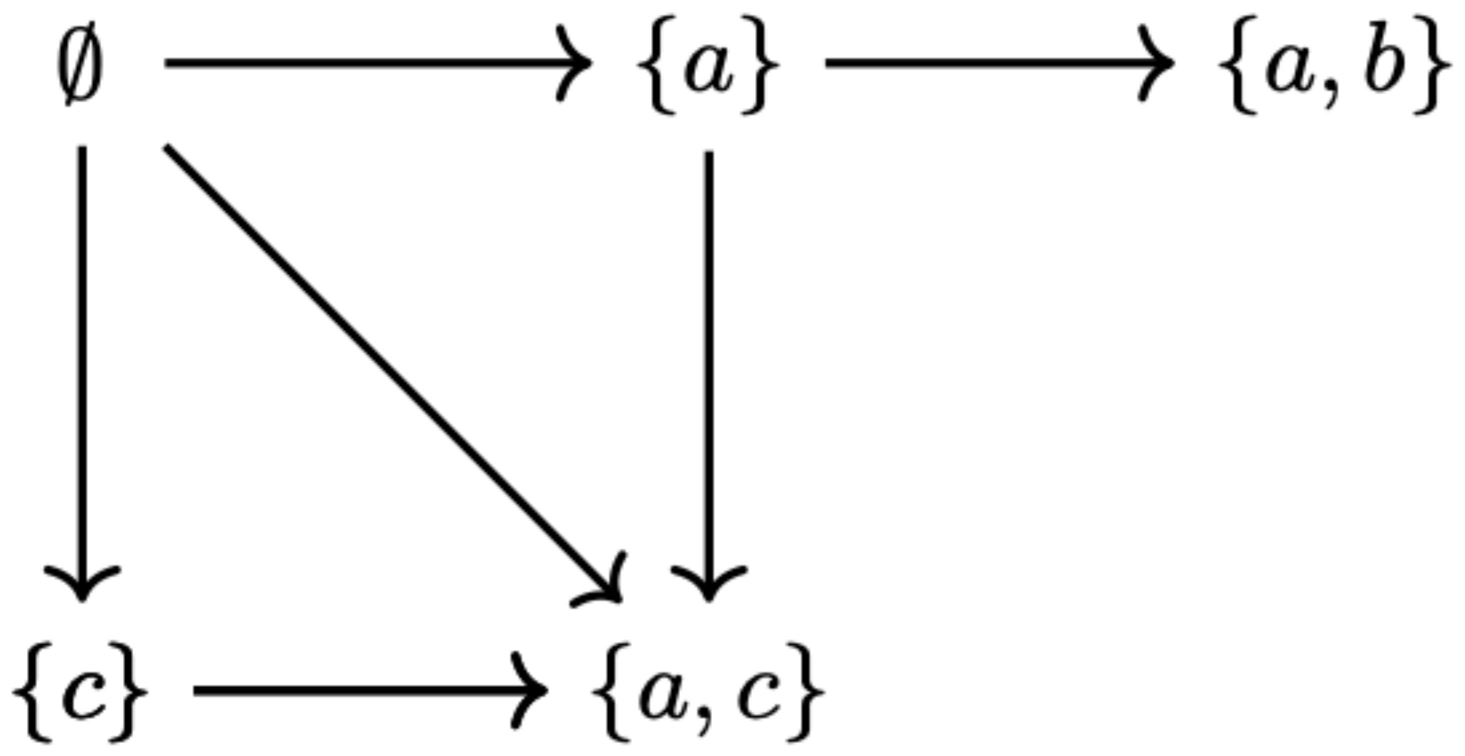


AESs

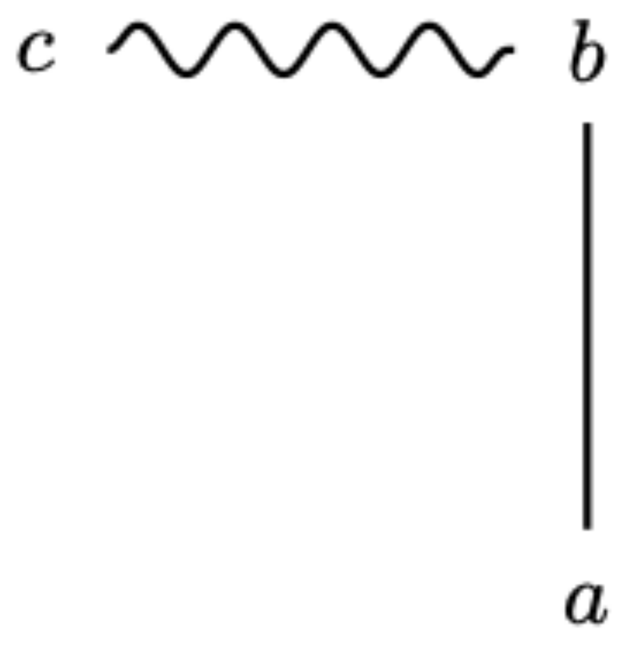




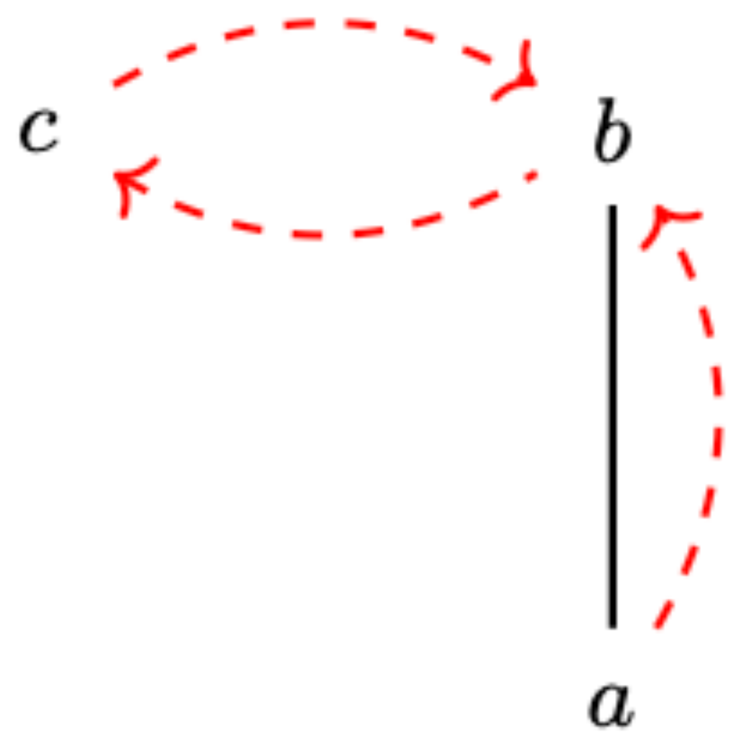
a < b   b#c



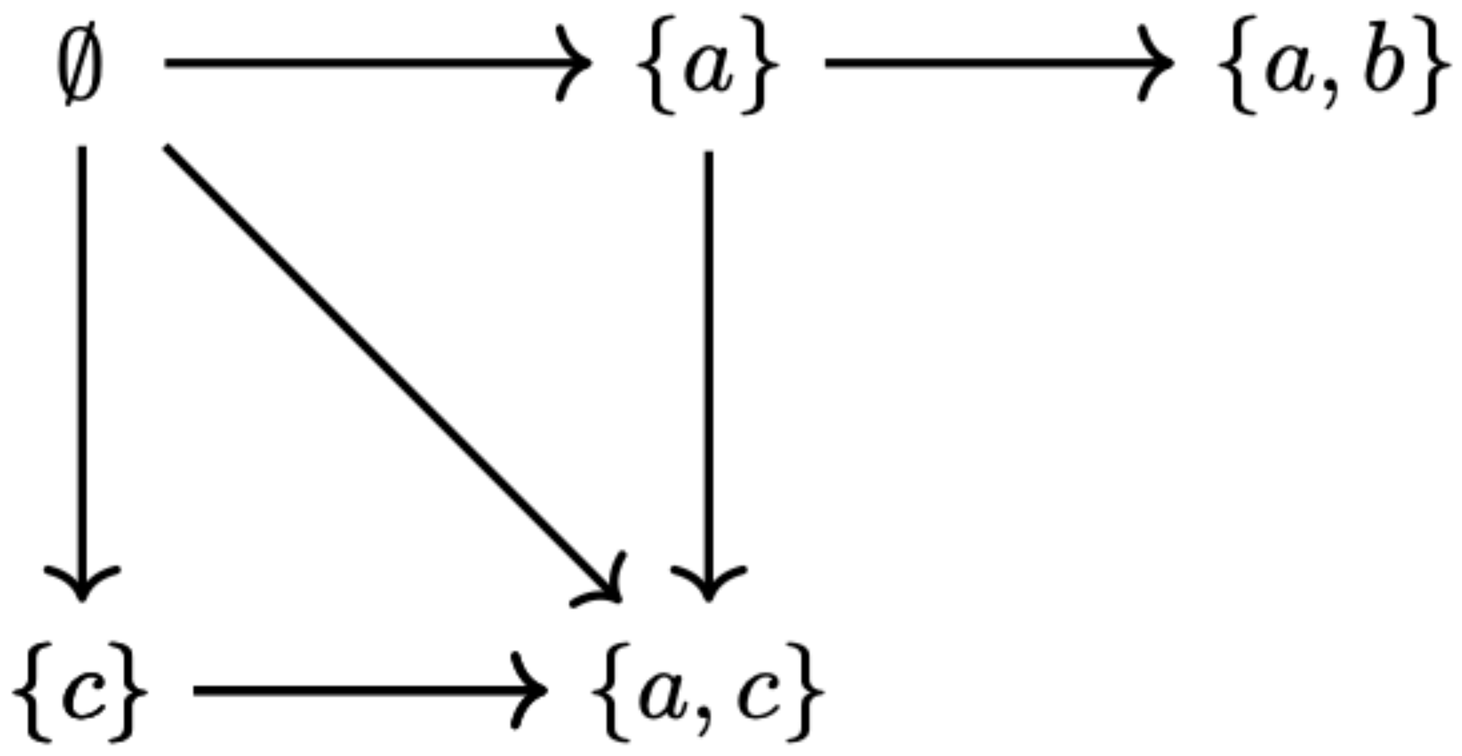
AESs



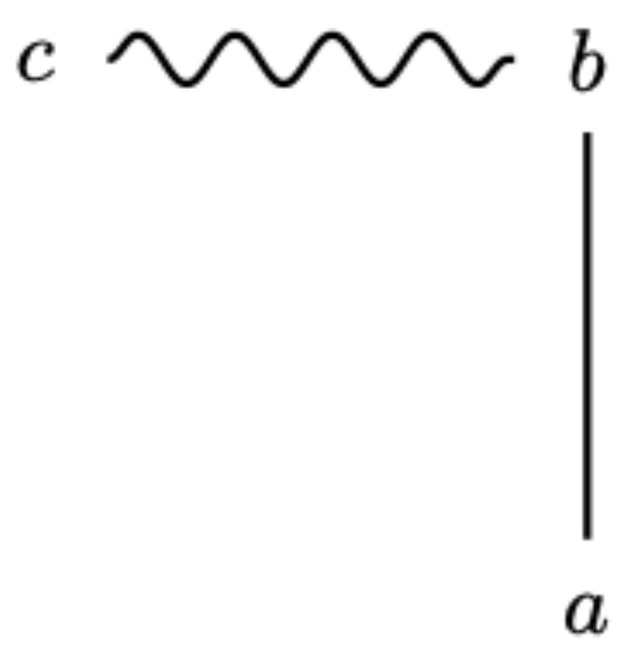
a < b   b#c



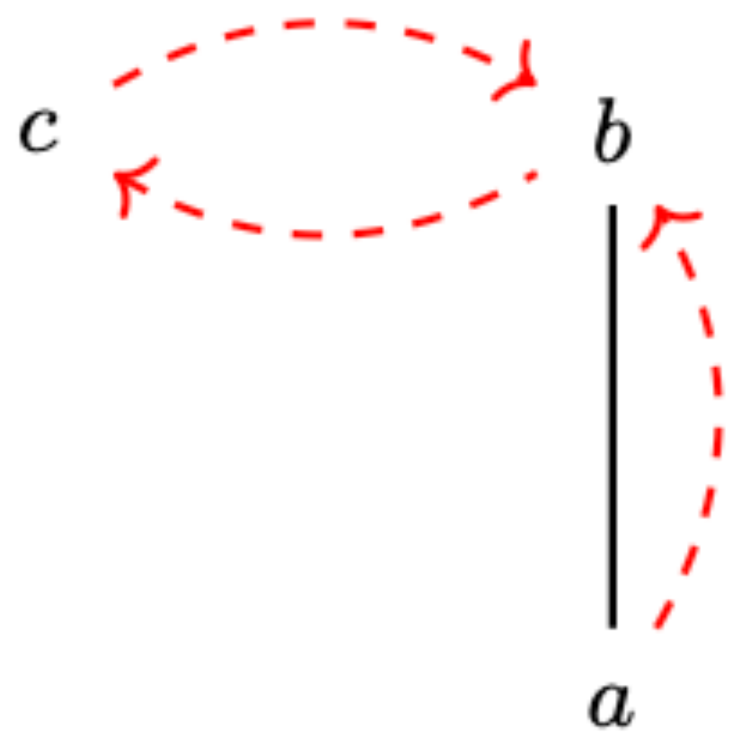
a < b   b ↗ c   c ↗ b



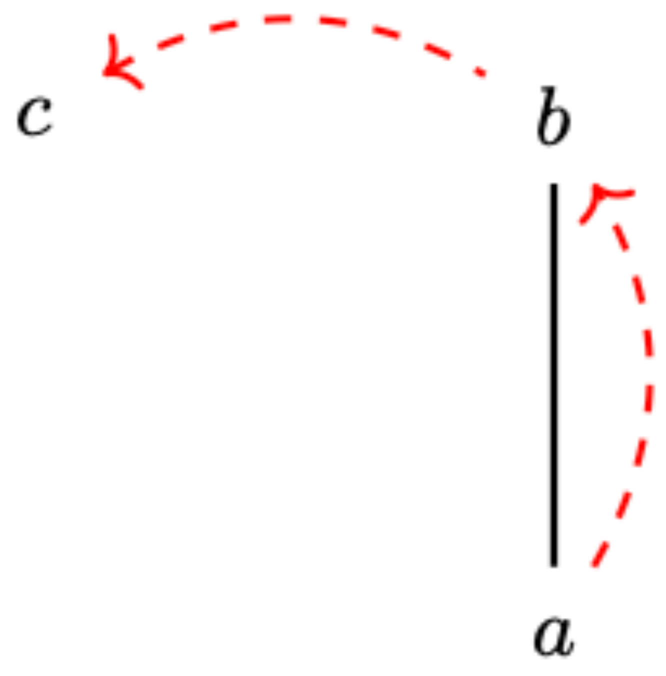
AESs

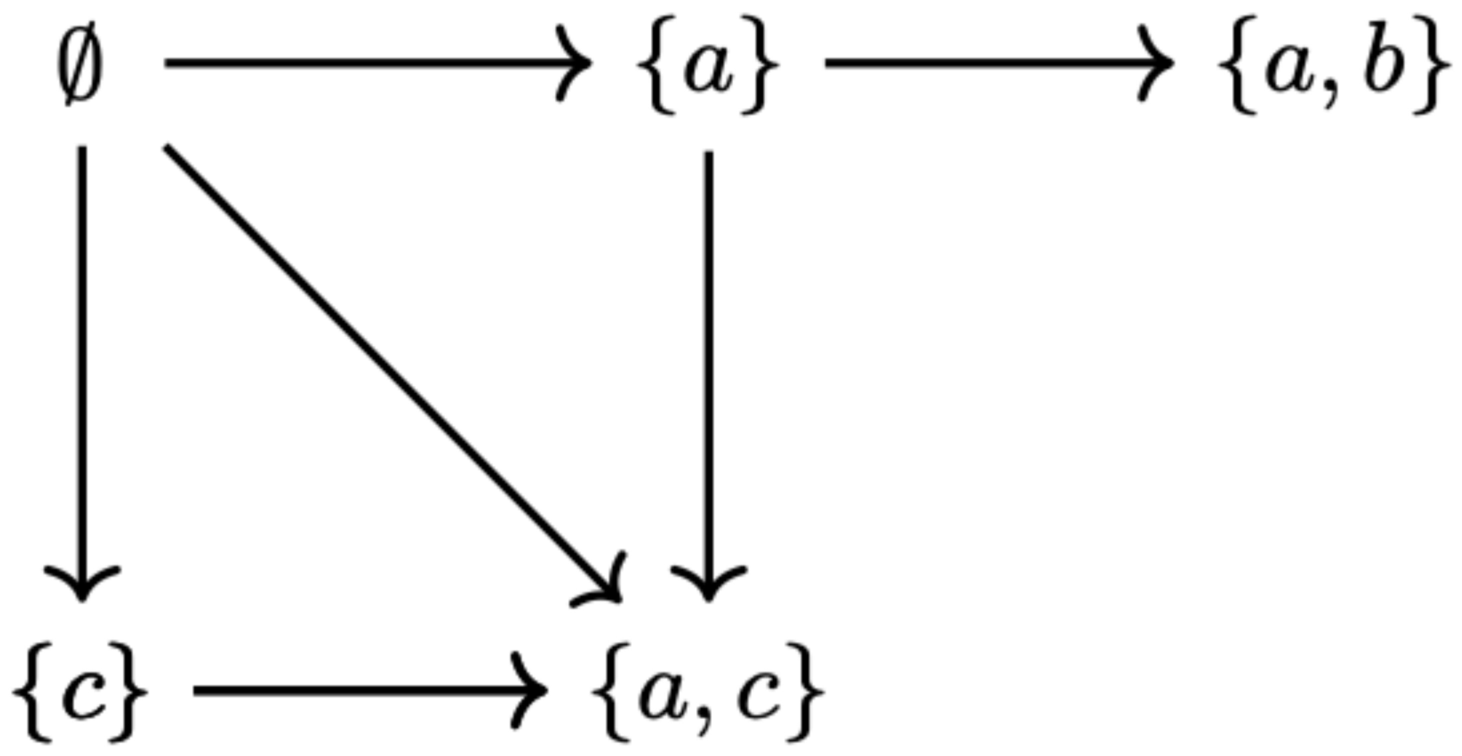


a < b   b#c

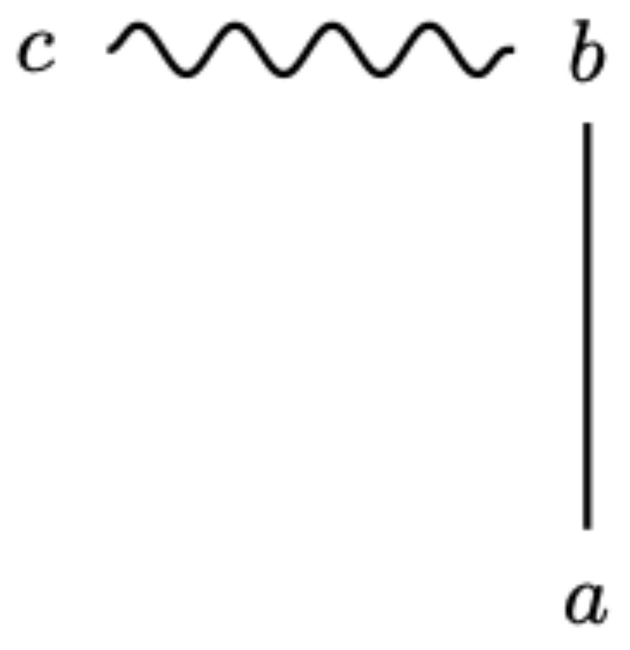


a < b   b $\nearrow$ c   c $\nearrow$ b

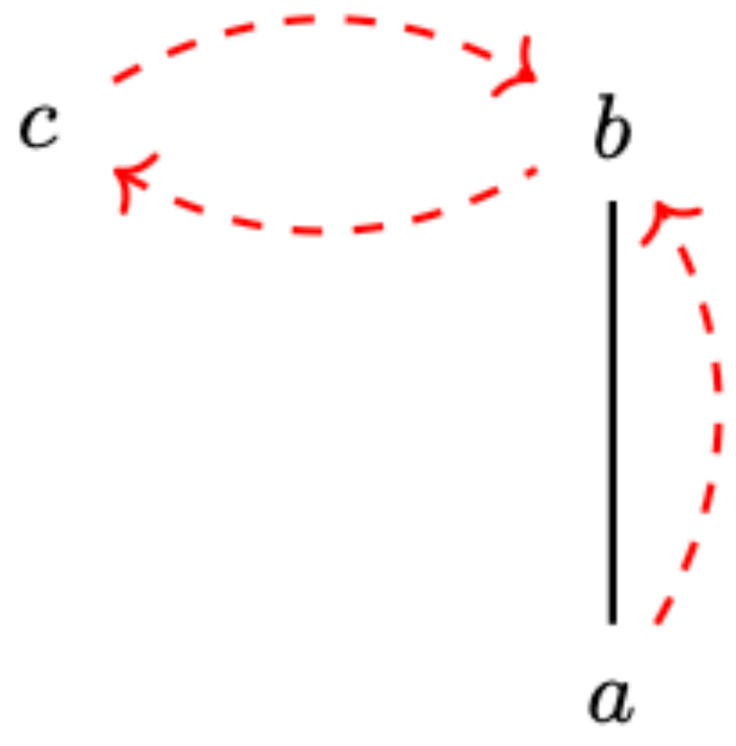




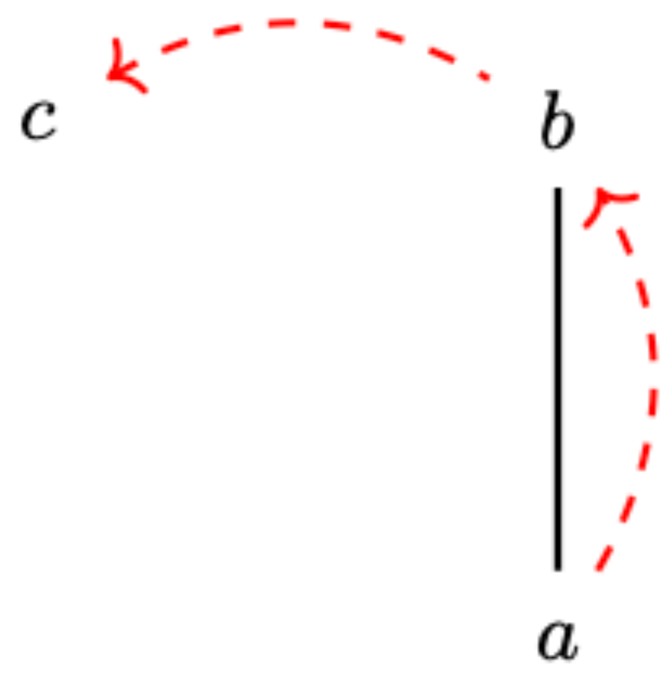
AESs



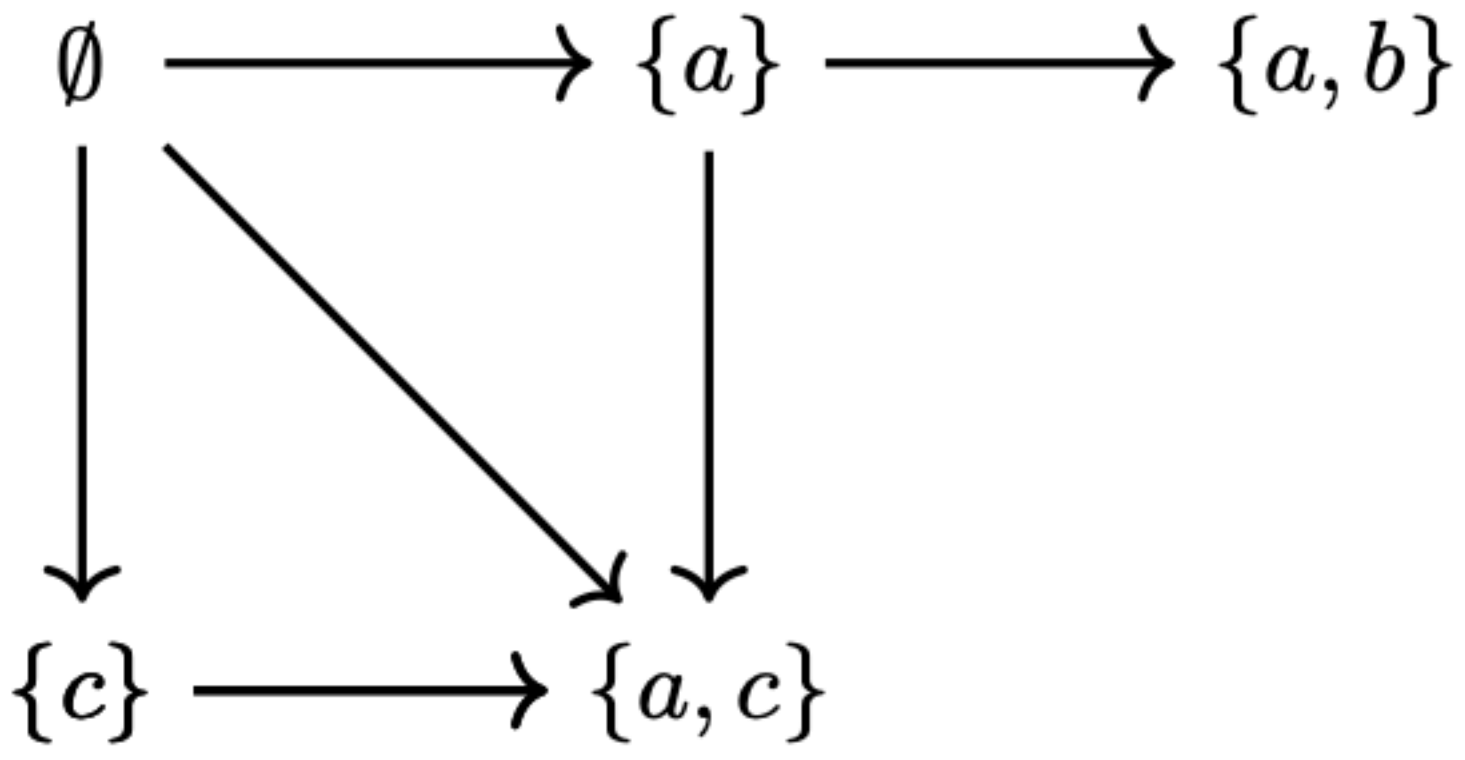
$a < b \quad b \# c$



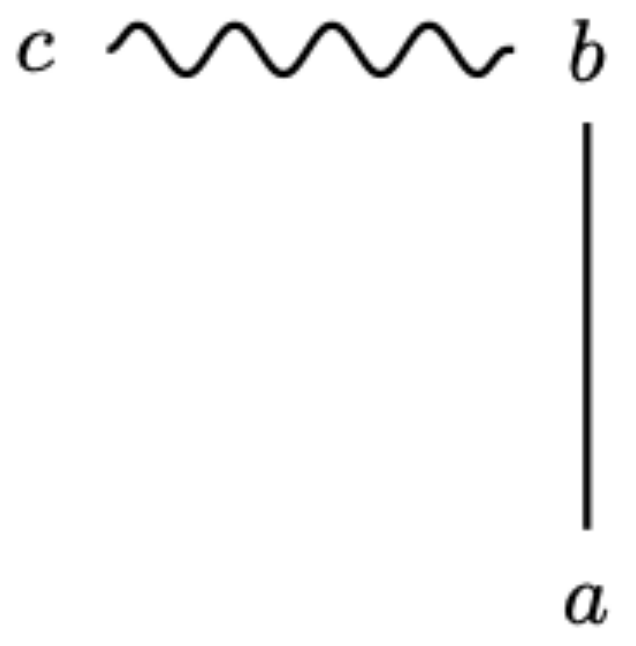
$a < b \quad b \nearrow c \quad c \nearrow b$



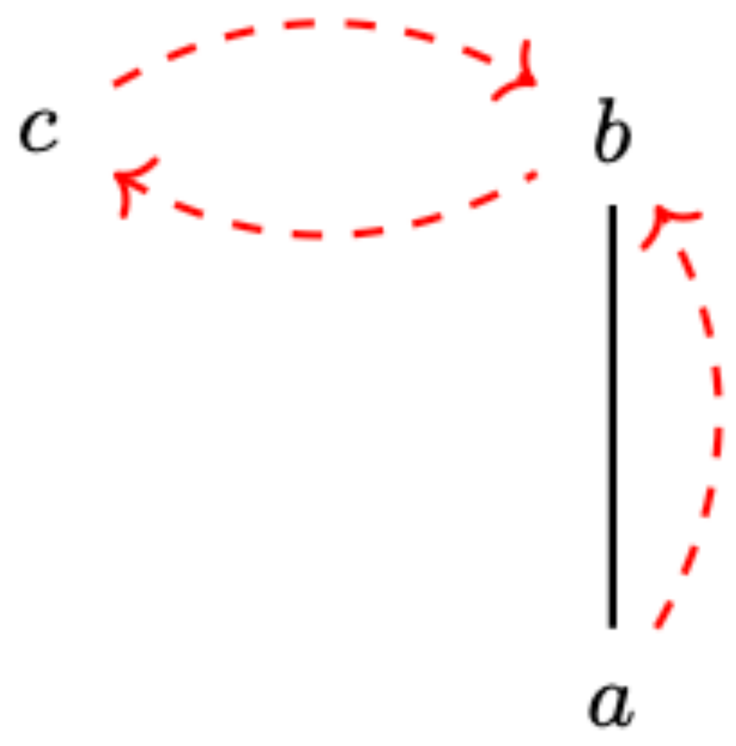
$a < b \quad b \nearrow c$



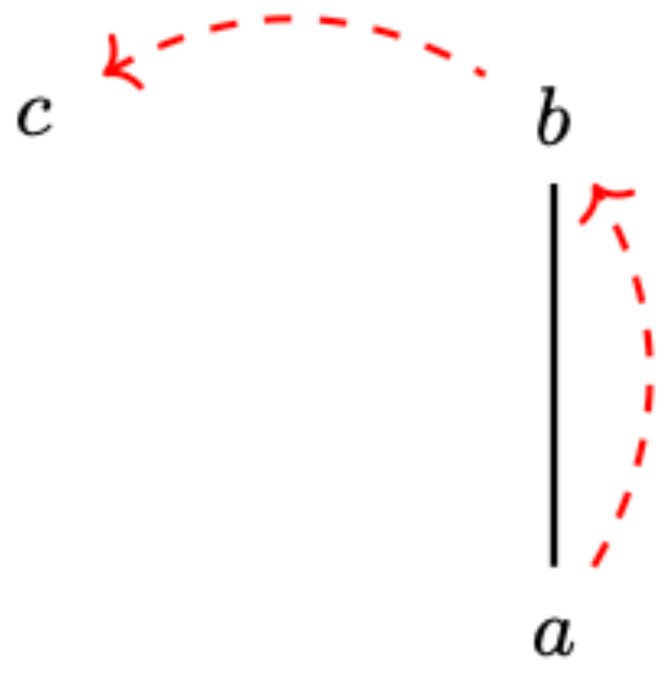
AESs



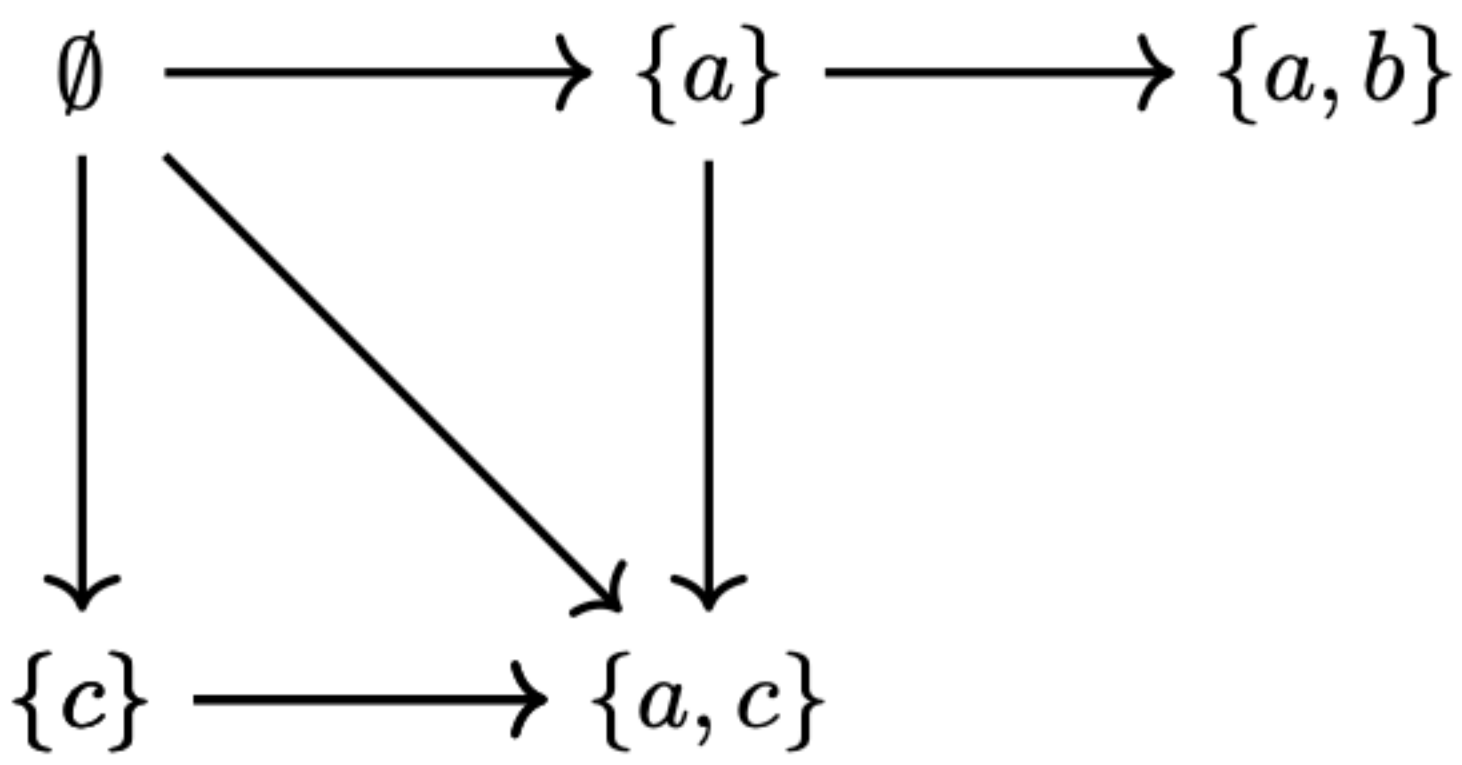
$a < b \quad b \# c$

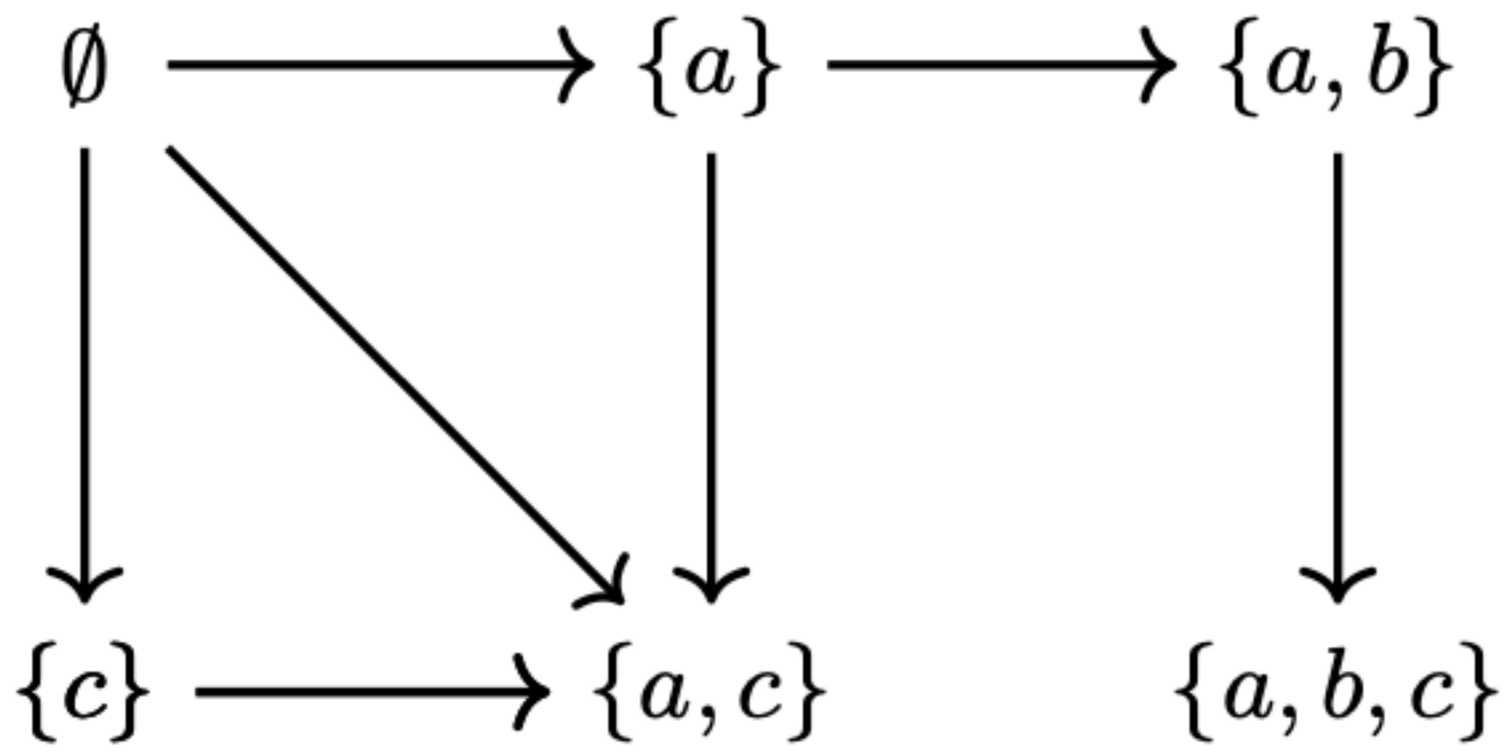


$a < b \quad b \nearrow c \quad c \nearrow b$

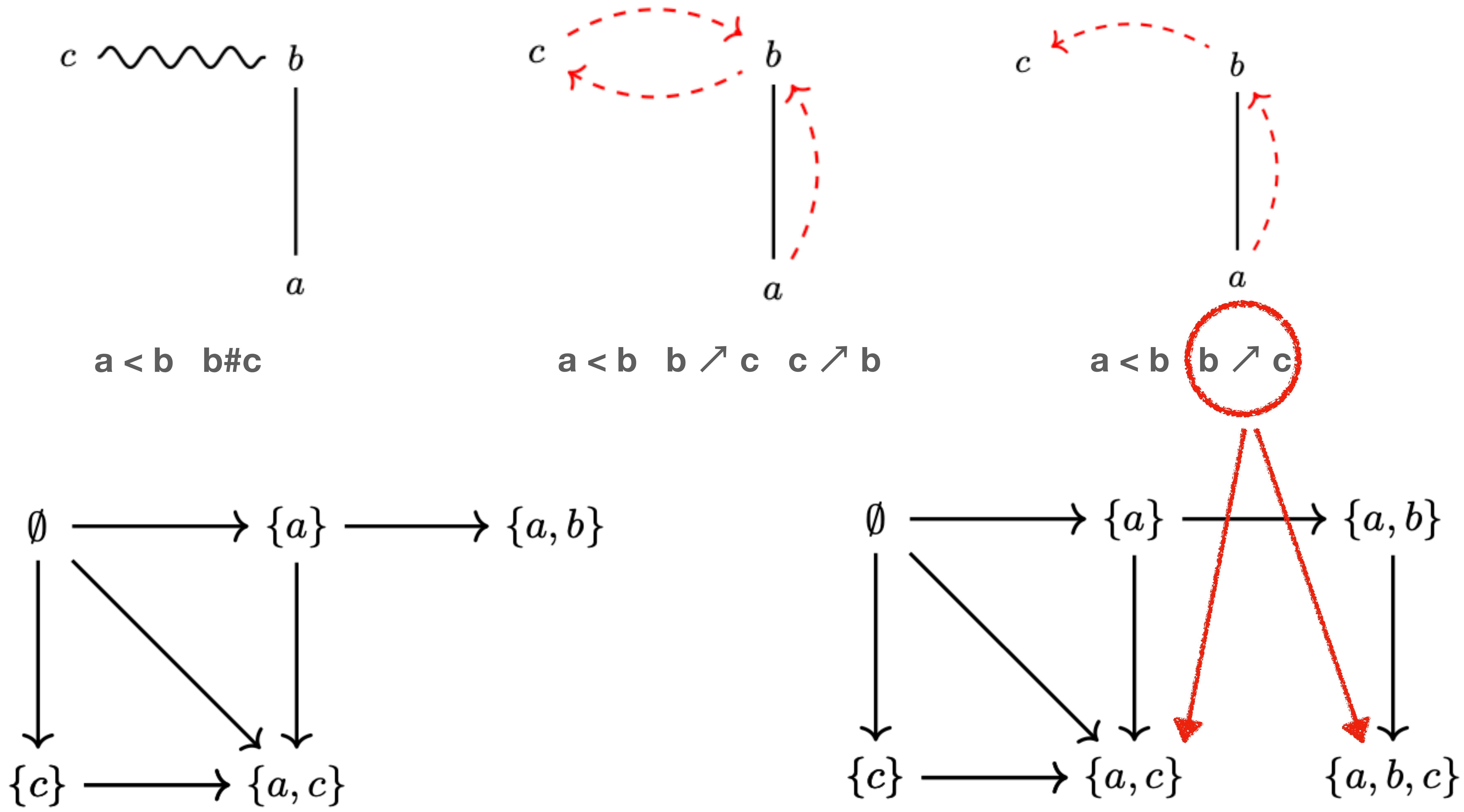


$a < b \quad b \nearrow c$

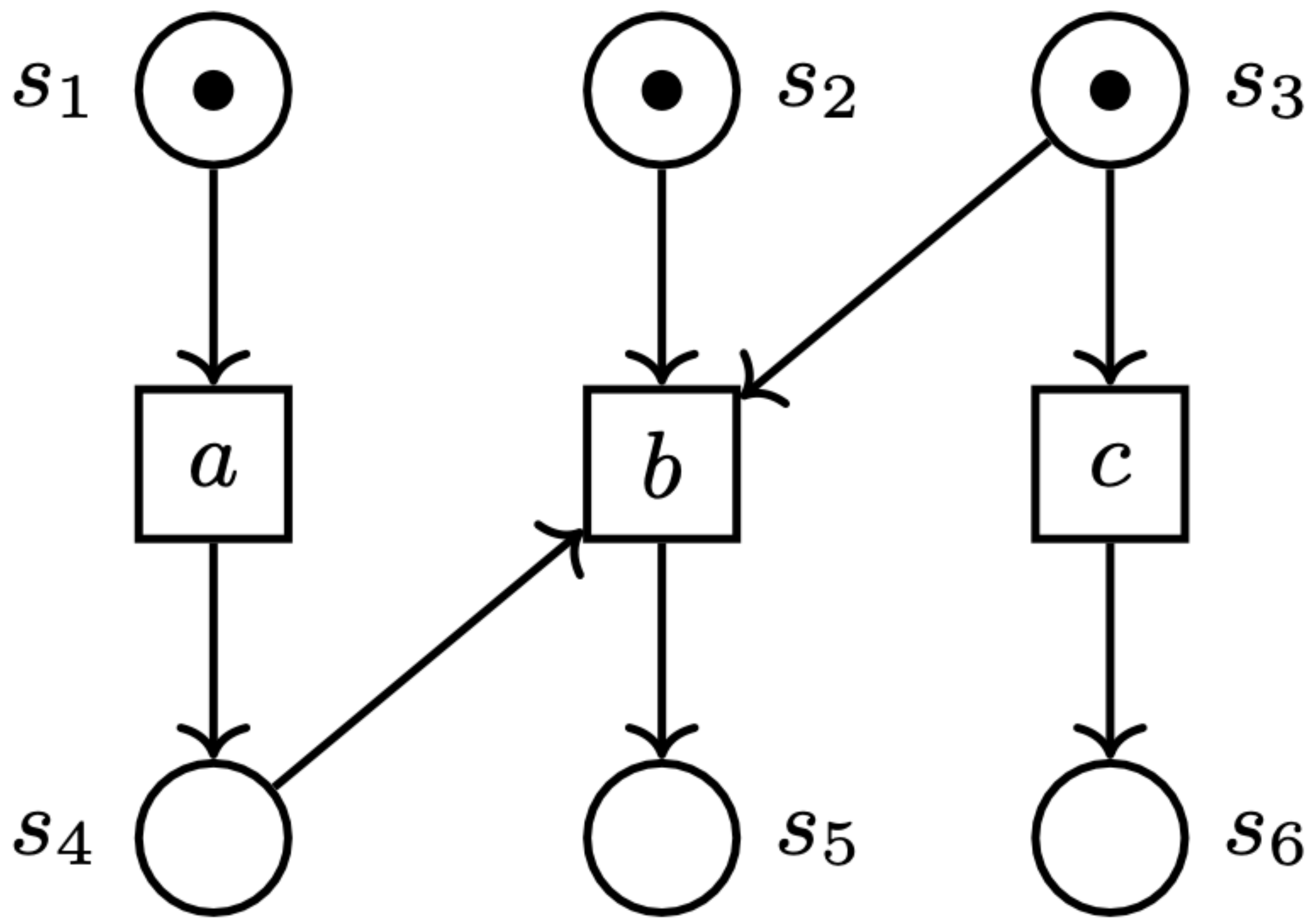




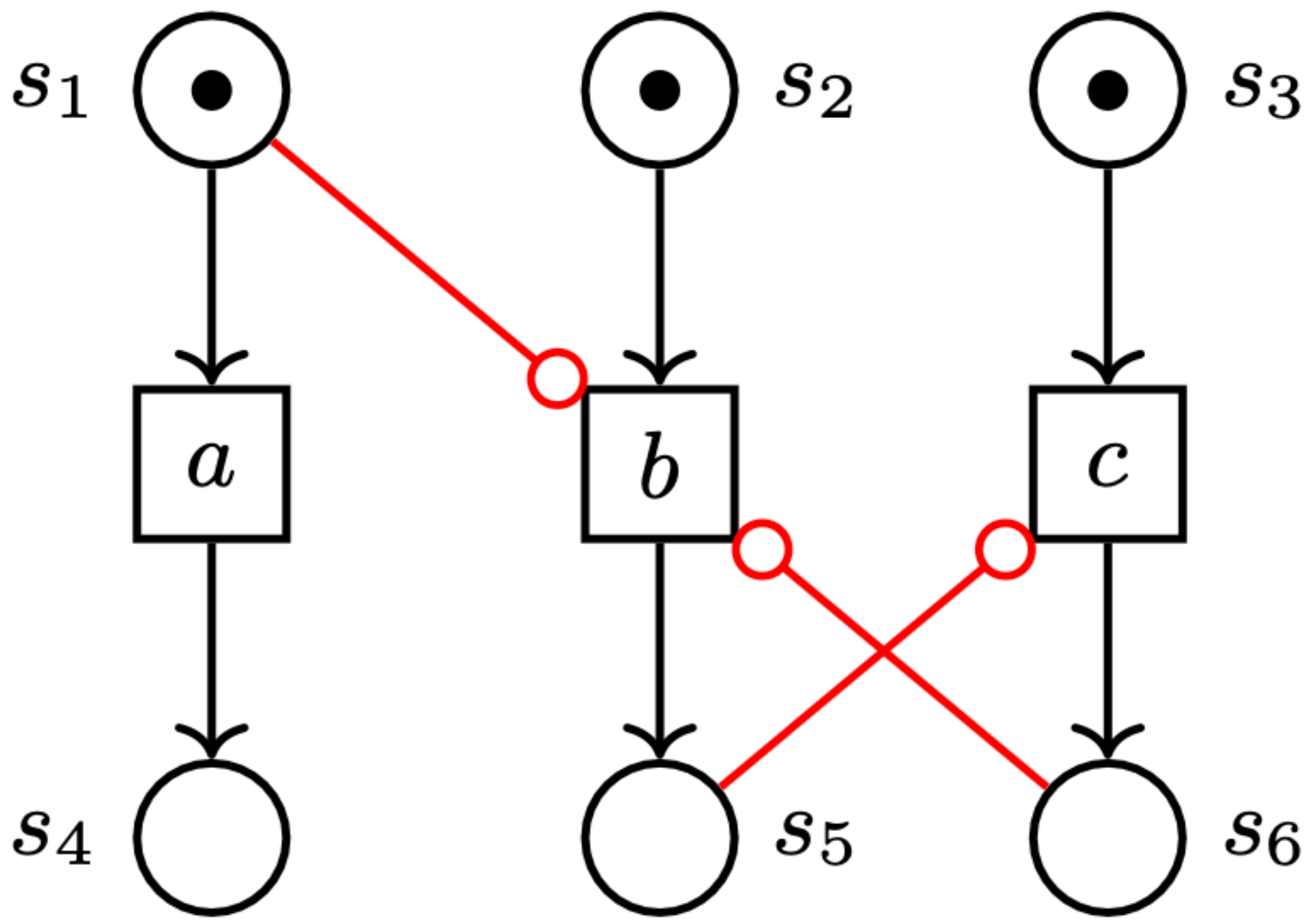
AESs



Reverse Asynchronous Causal Net



$a < b$ and $b \# c$



$a < b$, $b \nearrow c$ and $c \nearrow b$

Asynchronous conflicts?

```
pthread_mutex_t m = //initialization
int *x=malloc(sizeof(int));      ← event a
void thread(void *arg)
{   pthread_mutex_lock(&m);
    if(x != NULL)
      doSomething(x);    ← event b
    pthread_mutex_unlock(&m);
}
int main()
{   pthread_t t;
    pthread_create(&t, NULL, thread,
     NULL);
    pthread_mutex_lock(&m);
        free(x);   ← event c
    pthread_mutex_unlock(&m);
    return 0;}
```

a < b

a < c

c can happen after b

b cannot happen after c

$b \nearrow c$

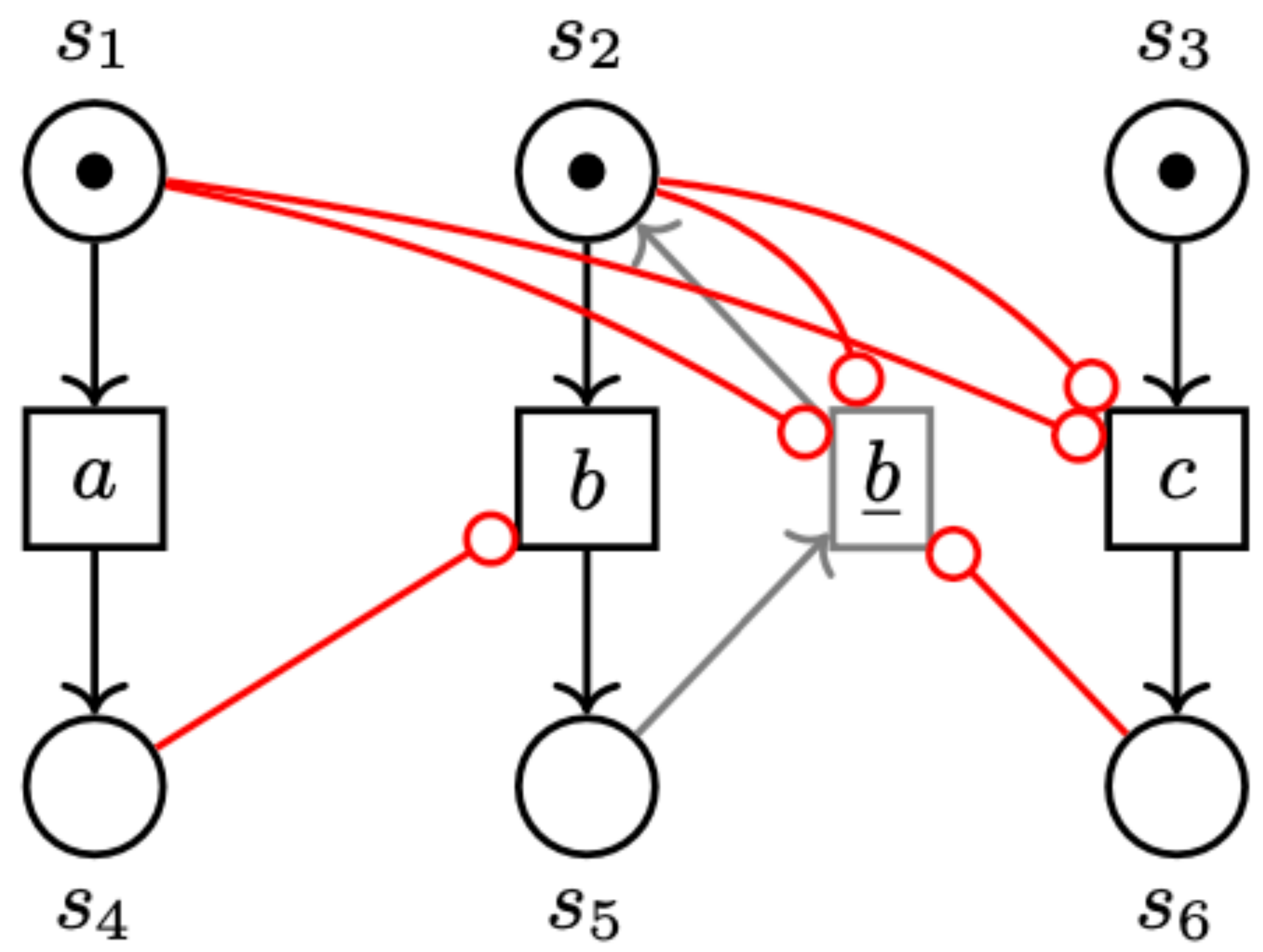
Reversing

```
pthread_mutex_t m = //initialization
int *x=malloc(sizeof(int));   ← event a
void thread(void *arg)
{    pthread_mutex_lock(&m);
     if(x != NULL)
       doSomething(x);   ← event b
     pthread_mutex_unlock(&m);
}
int main()
{    pthread_t t;
     pthread_create(&t, NULL, thread,
      NULL);
     pthread_mutex_lock(&m);
         free(x);   ← event c
     pthread_mutex_unlock(&m);
     return 0;}
```

c is reversed before
b is reversed

b ▷ c

From rCNs to rAESs



$E = \{a,b,c\}$

$U = \{\underline{b}\}$

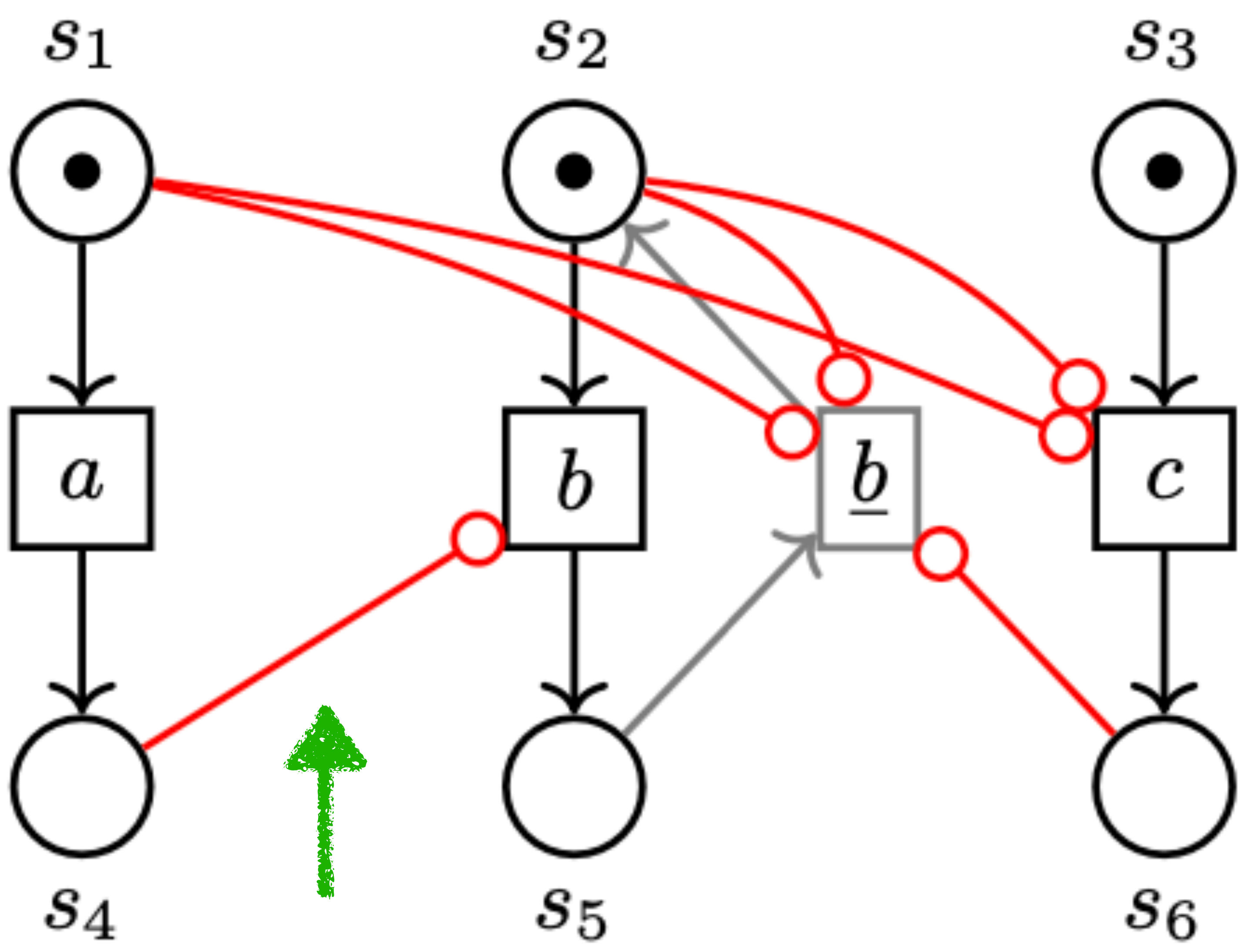
$< \; = \{(a,b), (b,c)\}$

$\nearrow \; = \{(a,c), (b,c), (b,a)\}$

$\prec \; = \{(a,\underline{b}), (\underline{b},\underline{b})\}$

$\triangleright \; = \{(\underline{b},c)\}$

From rCNs to rAESs



$E = \{a,b,c\}$

$U = \{\underline{b}\}$

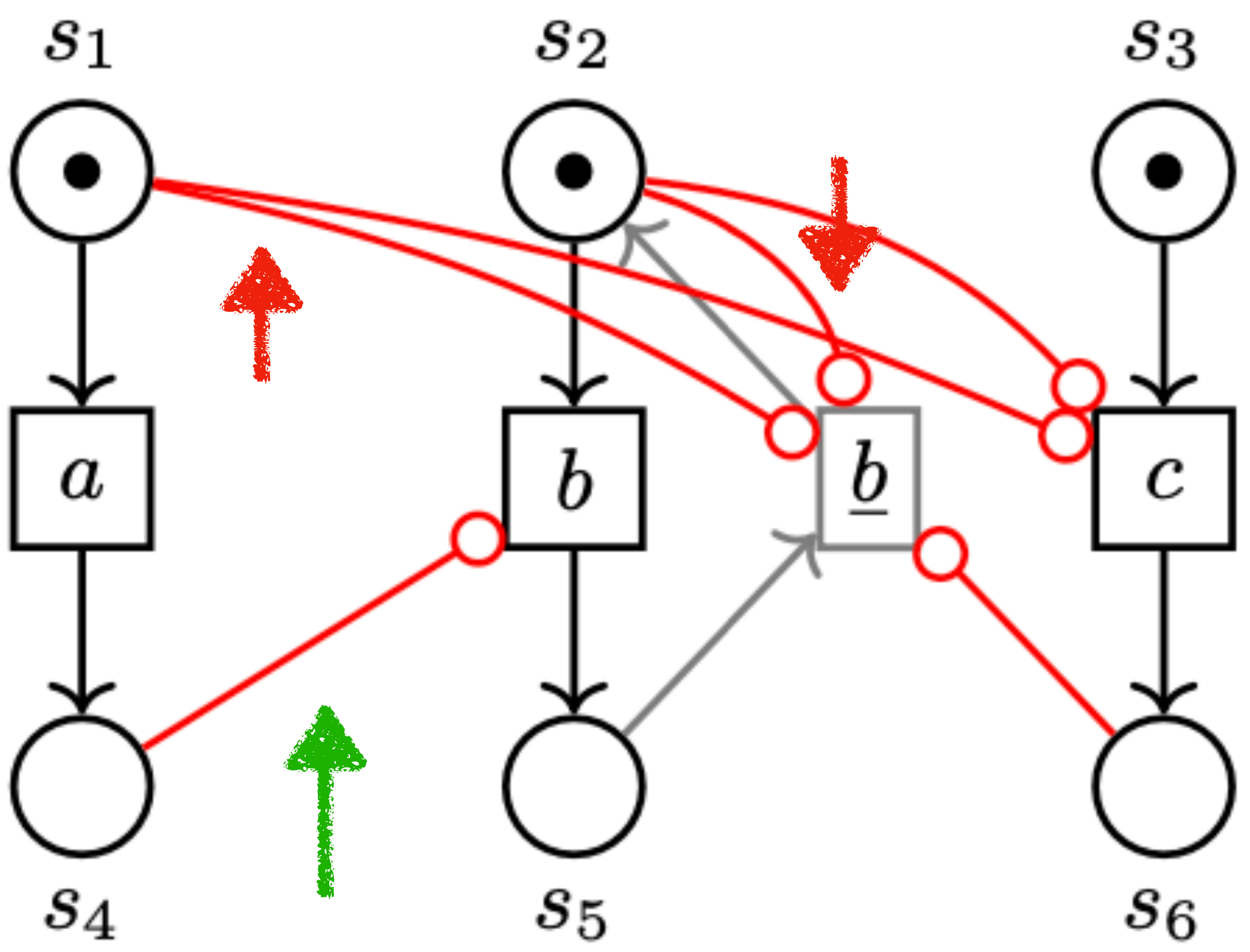
$< \; = \{(a,b), (b,c)\}$

$\nearrow \; = \{(a,c), (b,c), (b,a)\}$

$< \; = \{(a,\underline{b}), (\underline{b},\underline{b})\}$

$\triangleright \; = \{(\underline{b},c)\}$

From rCNs to rAESs



$E = \{a,b,c\}$

$U = \{\underline{b}\}$

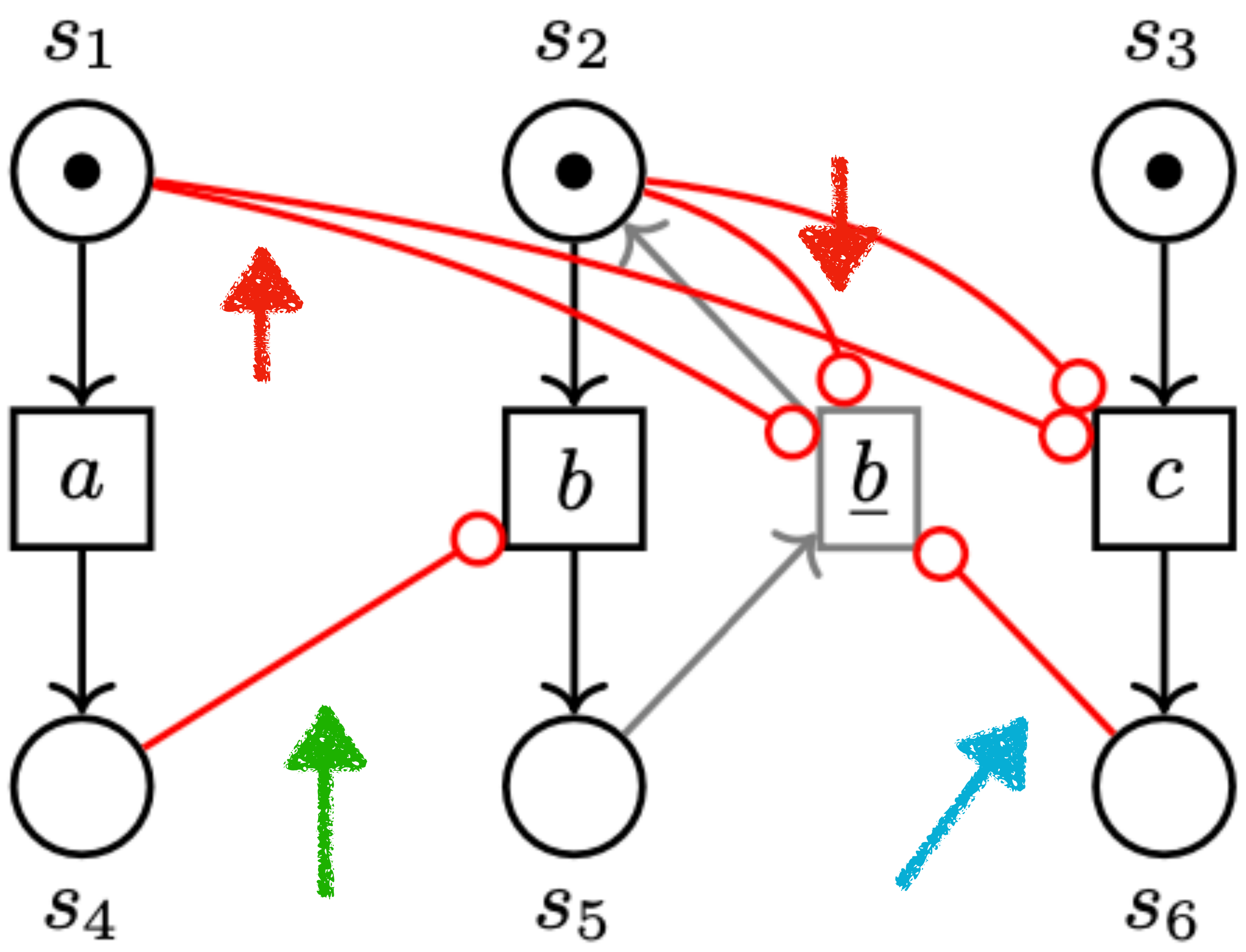
$< \; = \{(a,b), (b,c)\}$

$\nearrow \; = \{(a,c), (b,c), (b,a)\}$

$< \; = \{(a,\underline{b}), (\underline{b},\underline{b})\}$

$\triangleright \; = \{(\underline{b},c)\}$

From rCNs to rAESs



E = {a,b,c}

U = {$\underline{b}$}

$<$ = {(a,b), (b,c)}

$\nearrow$ = {(a,c), (b,c), (b,a)}

$\prec$ = {(a,$\underline{b}$), ($\underline{b}$,$\underline{b}$)}

$\triangleright$ = {($\underline{b}$,c)}

Results (correspondence)

Theorem 1. *Let $V^{\underline{T}}$ be an r*ACN. *Then $\mathcal{E}_r(V)$ is an r*AES.

Theorem 2. *Let* $\mathsf{H} = (E, U, <, \nearrow, \prec, \lhd)$ *be an r*AES. *Then $\mathcal{N}_r(\mathsf{H})$ is an r*ACN.

Also, we can show a correspondence in terms of configurations

Let H an rAES. Then $X \in \mathbf{Conf}(\mathsf{H})$ iff $X \in \mathbf{Conf}(\mathcal{N}_r(\mathsf{H}))$

Let $V^{\underline{T}}$ an rACN. Then $X \in \mathbf{Conf}(V^{\underline{T}})$ iff $X \in \mathbf{Conf}(\mathcal{E}_r(V))$

Results (categories)

Proposition 1. *$\mathcal{E}_r$: **RACN** $\to$ **RAES** is a well-defined functor.*

Proposition 3. *$\mathcal{N}_r$: **RAES** $\to$ **RACN** is a well-defined functor.*

Theorem 3. *The functor $\mathcal{N}_r$: **RAES** $\to$ **RACN** is the left adjoint of the functor $\mathcal{E}_r$: **RACN** $\to$ **RAES**.*

Theorem 4. *The functor $\mathcal{N}$: **AES** $\to$ **ACN** is the left adjoint of the functor $\mathcal{E}$: **ACN** $\to$ **AES**.*

Conclusions

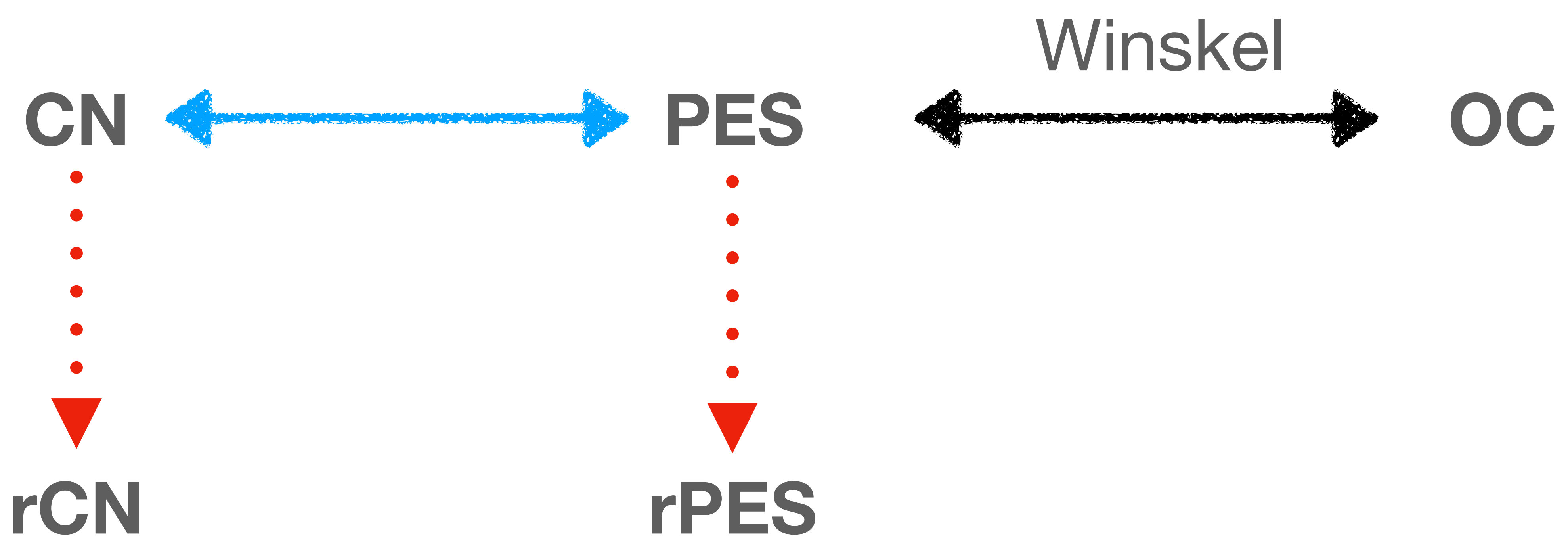
- We have established a correspondence between two different models
  - (Reversible) CNs and (reversible) AESs
- On the net side, all the relations are homogeneously modelled via inhibitor arcs
  - Inhibitor arcs are powerful enough

The picture

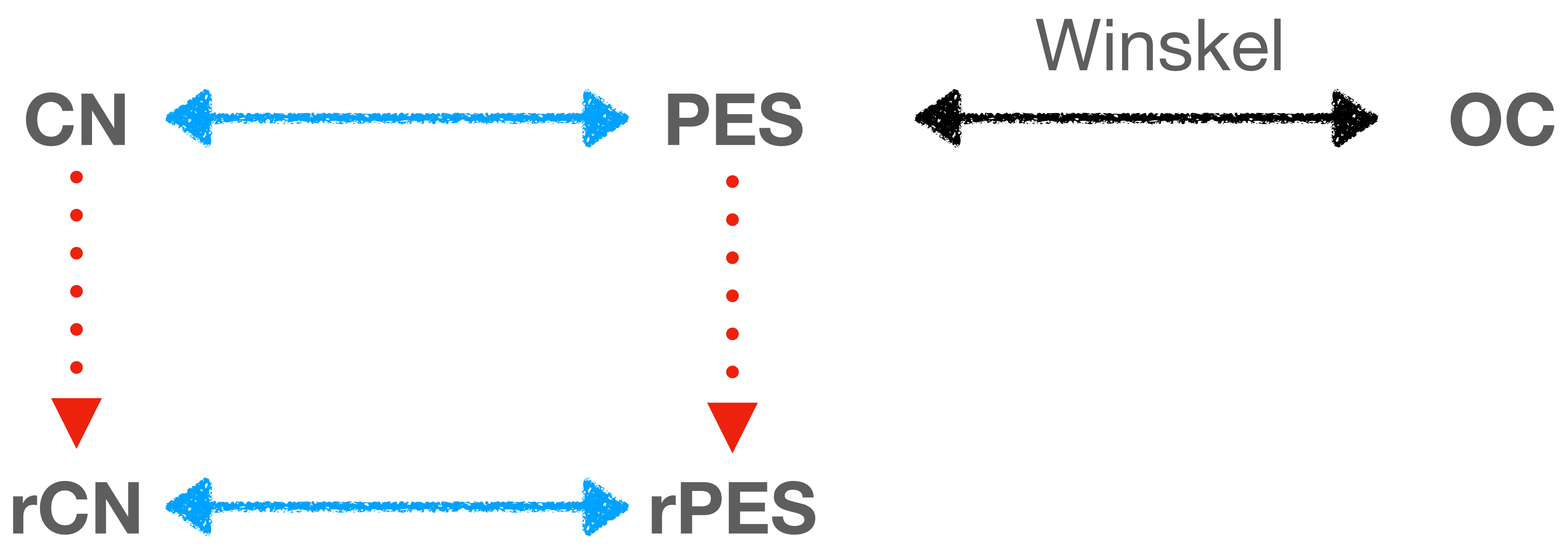
The picture

The picture

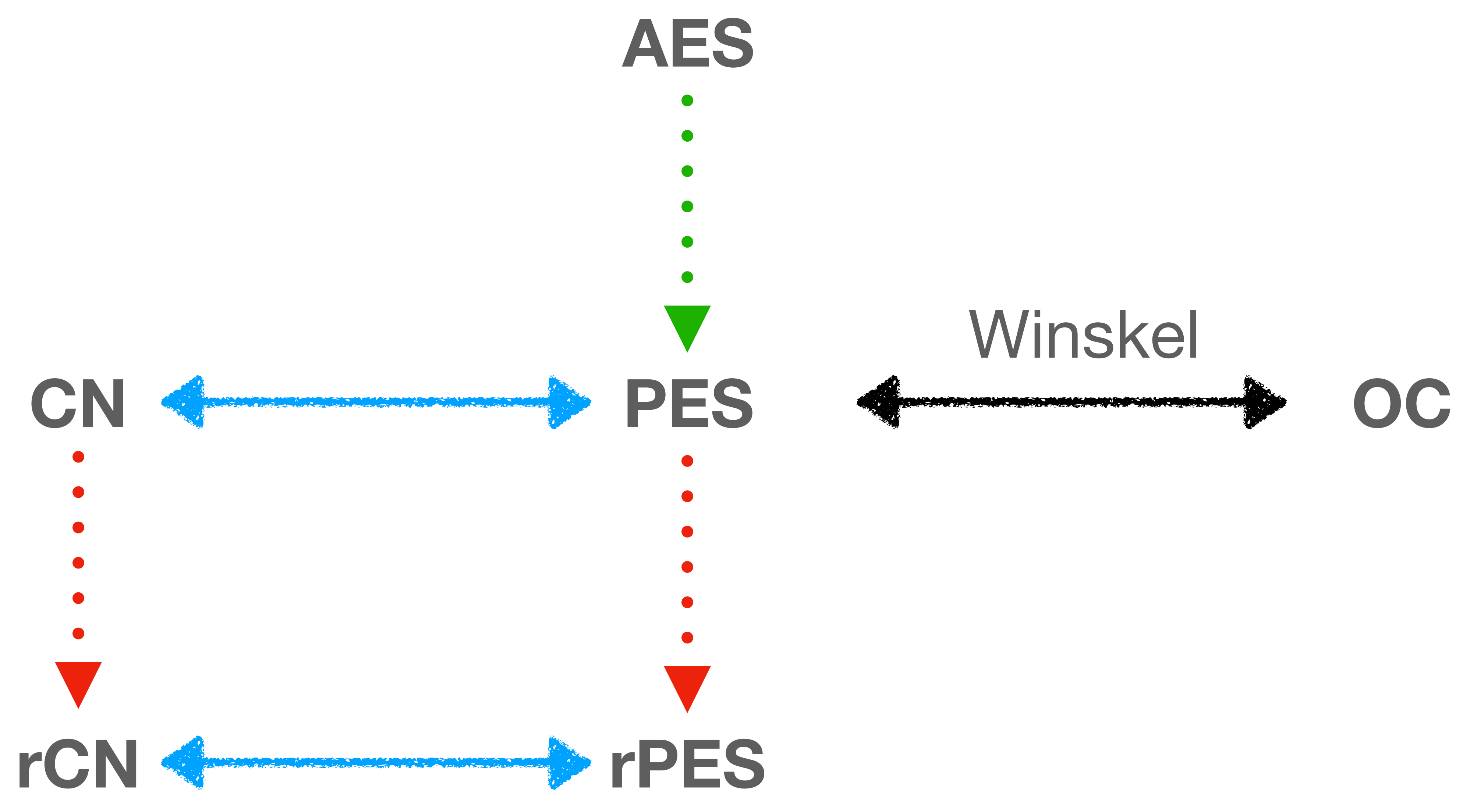
The picture



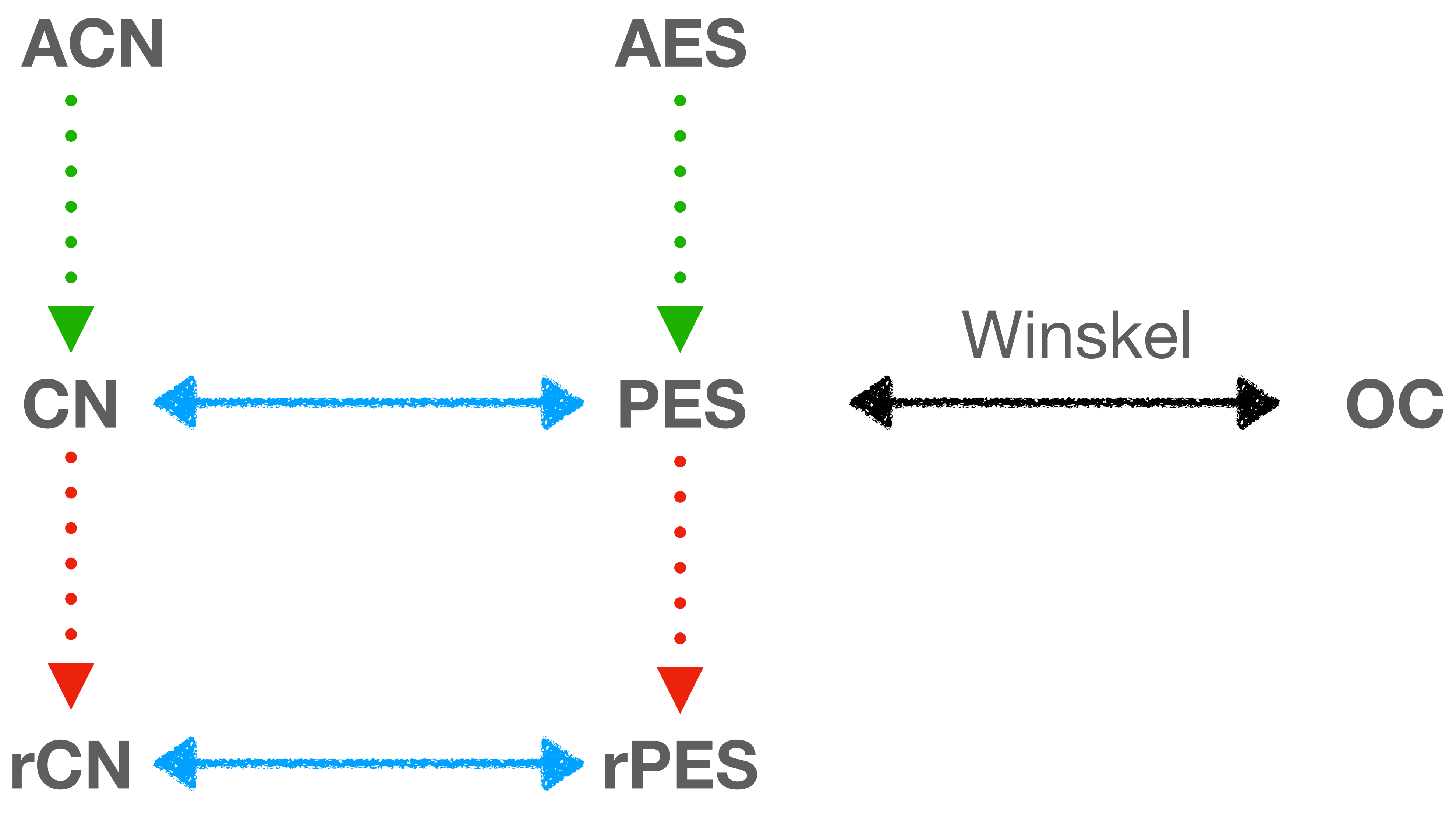
The picture



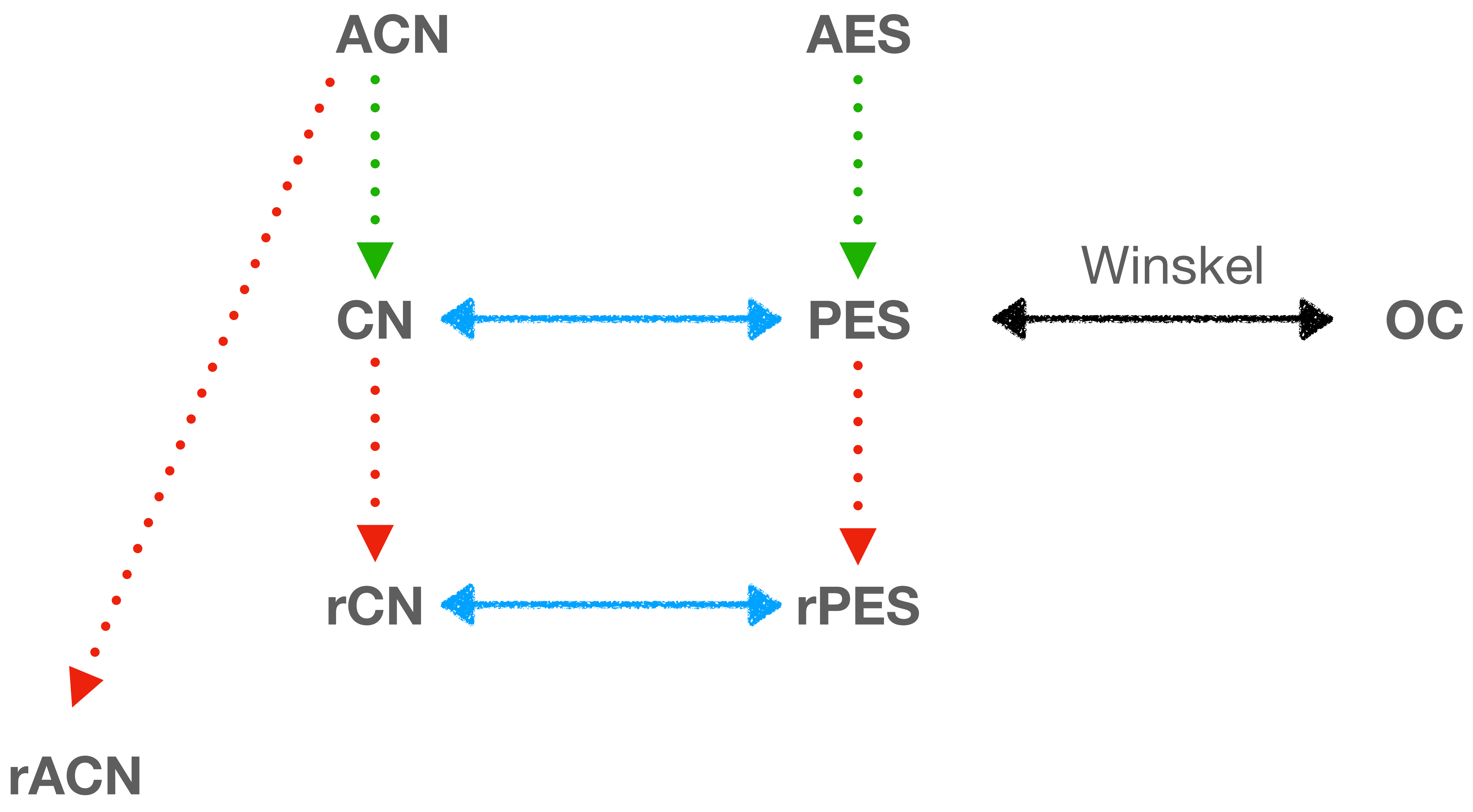
The picture



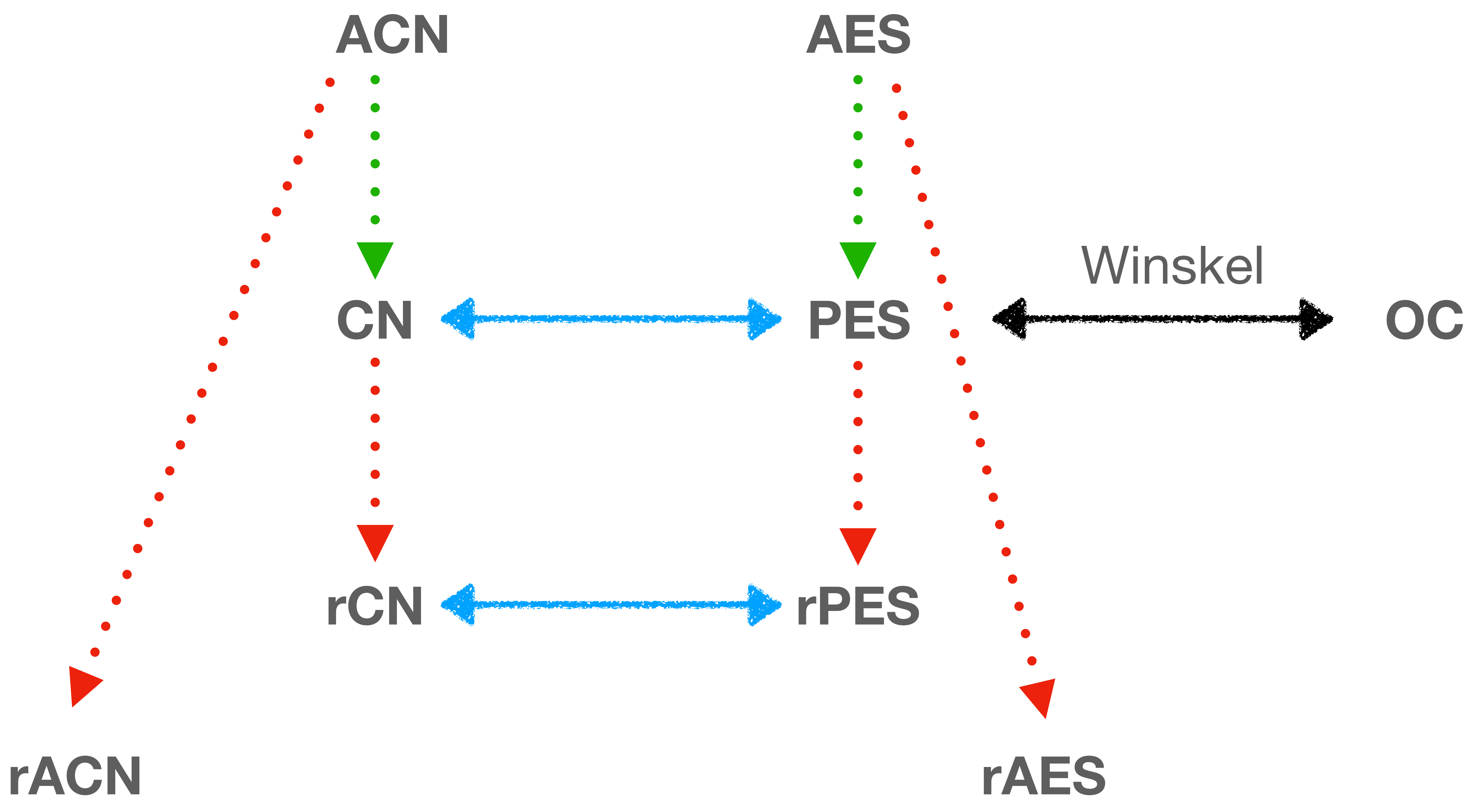
The picture



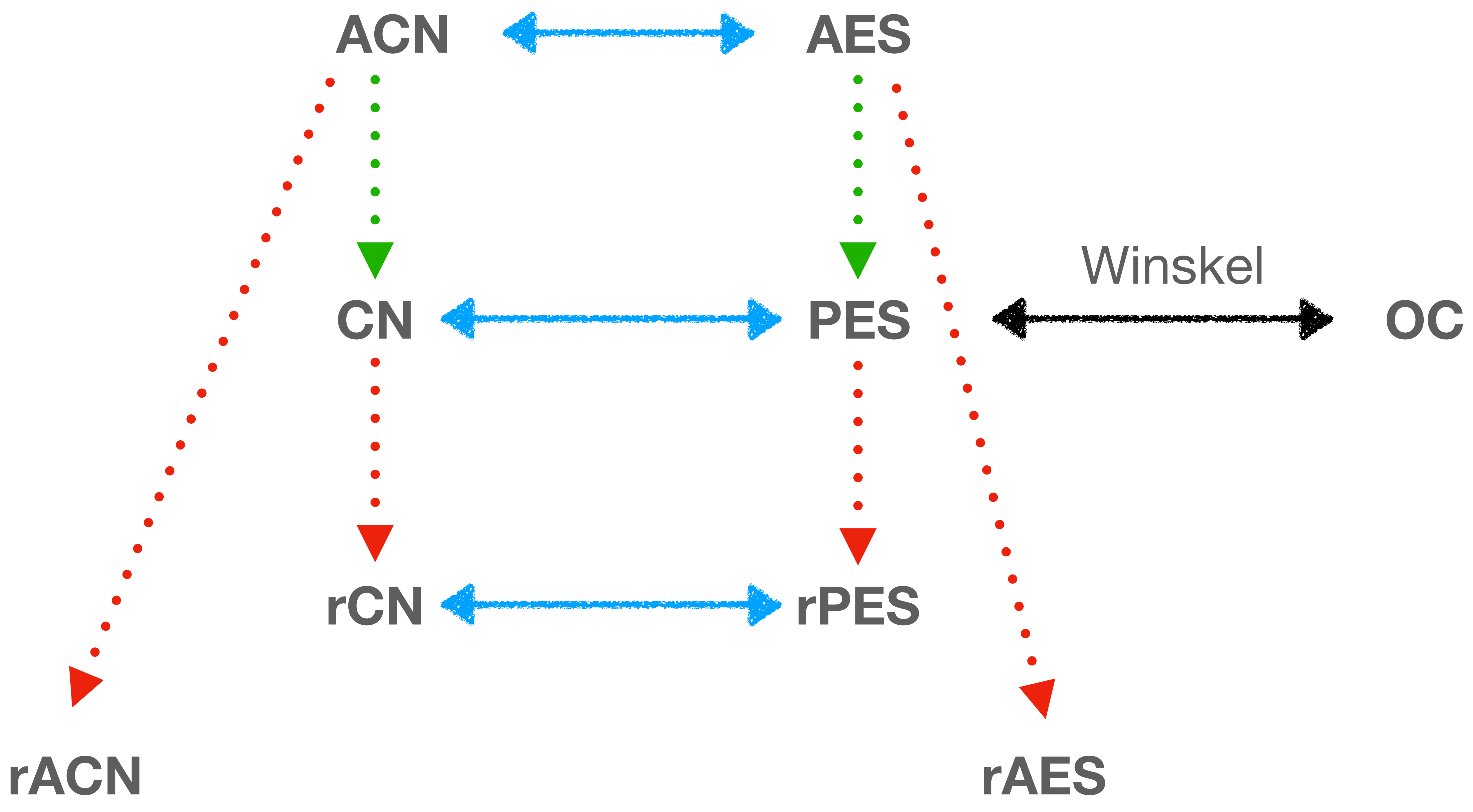
The picture



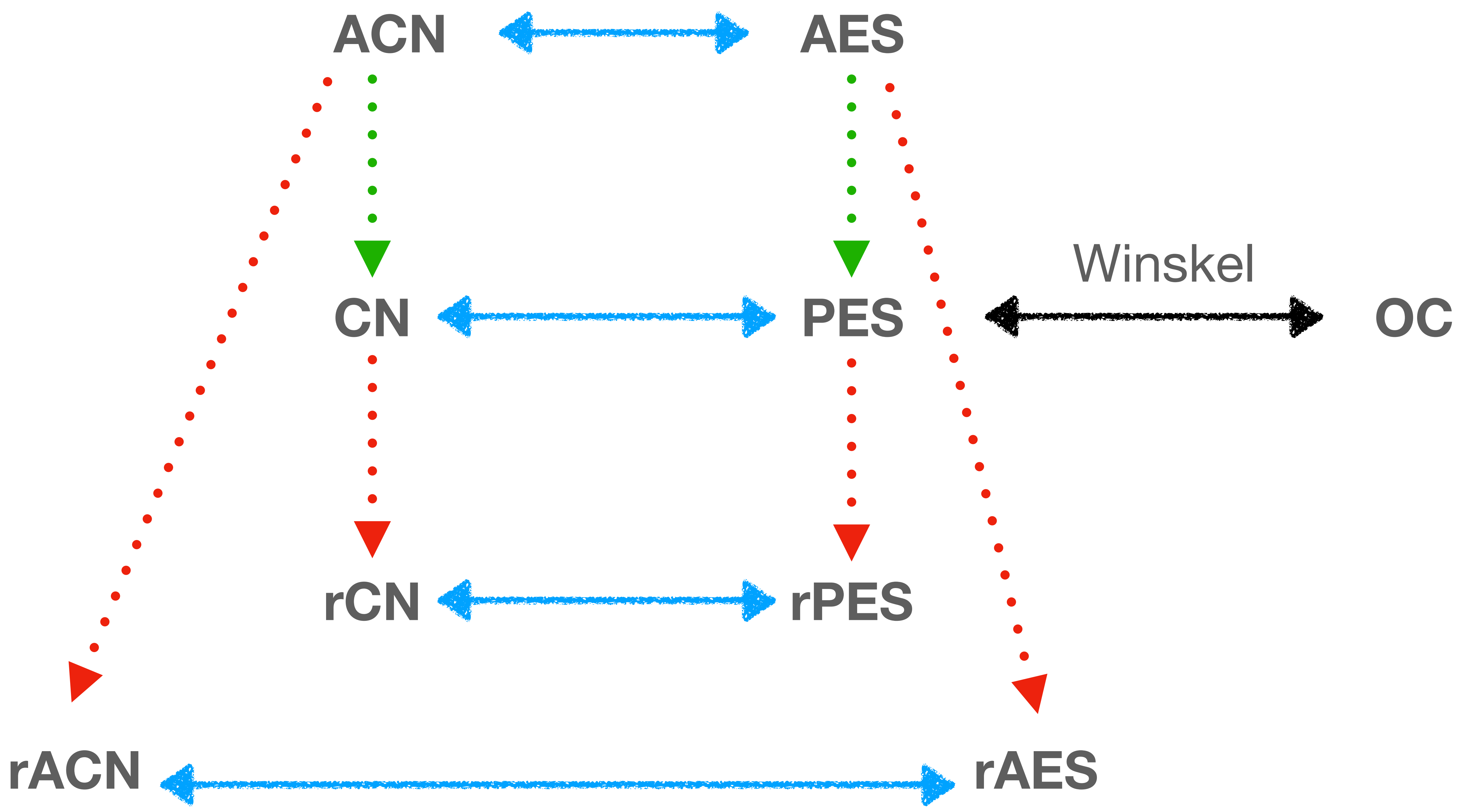
The picture



The picture



The picture



Future work

- The tight correspondence between rCNs and rAESs can be exploited in debugging
  - rAES can be used to give “constraints” to the system (e.g., express the desired behaviour)
  - rCNs can be used as the “operational” counterpart to be executed / reflected in the debugger
- Investigate which token philosophy obeys or rACN
  - Individual or collective?