Semantics for consistent activation in context-oriented systems

Florian Thuin

Ecole Polytechnique de Louvain

January 18, 2017

Information about the paper

Semantics for consistent activation in context-oriented systems

Nicolás Cardozo
Sebastián González
Kim Mens
Ragnhild Van Der Straeten
Jorge Vallejos
Theo D'Hondt

October 1, 2014

Find the full-text PDF at
`https://www.researchgate.net/publication/267634206`

Plan

1. Introduction

2. Context-oriented programming

3. Context modelling

4. Conclusion

Introduction

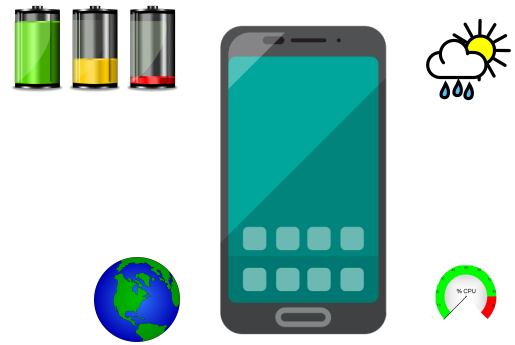
"Semantics for consistent activation in context-oriented systems"

What I will talk about:

- What is a context ?
- What is a context-oriented system ?
- How to model **context (de)activations** ?
- How to model **relations between contexts** ?

Hardware evolution & programming techniques

- Smartphones have a lot of ***sensors***, allowing programs to interact with their environment.



- You can use them to provide a better user experience
- Most of programming techniques don't provide a reusable and maintainable approach to do it

- Context-oriented programming has been made for this!

Plan

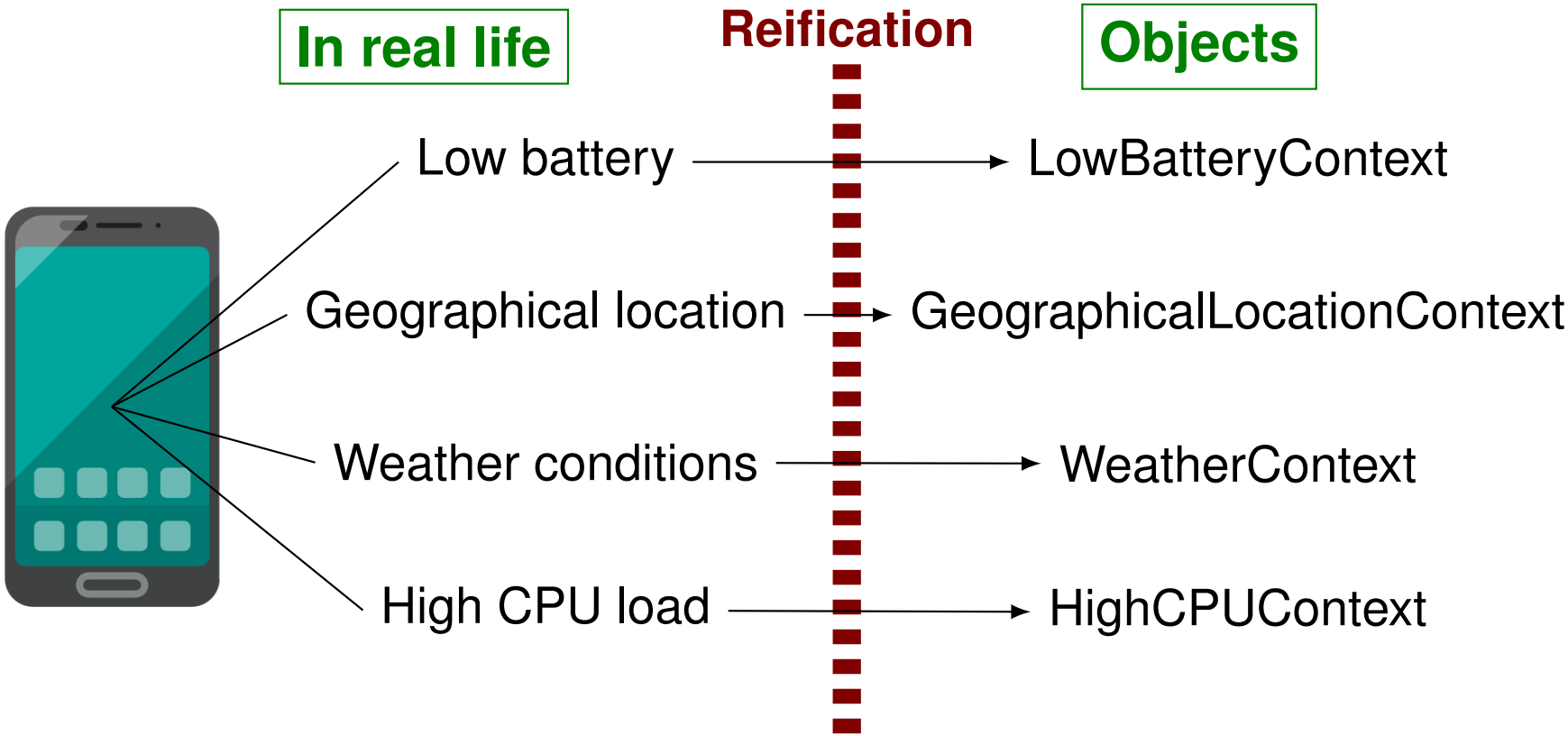
1. Introduction

2. Context-oriented programming

3. Context modelling

4. Conclusion

What is a context?



- The environment is modelled by contexts that are
  - active *or*
  - inactive
- This paradigm is called **context-oriented programming**

What is a context-oriented system?

A context-oriented system:

- is implemented with **context-oriented programming**
- has **Context** abstraction encoded as **first-class entities**
- has a certain number of *active* and *inactive* contexts
- has a **context-specific behavior** (depends on active contexts and their adaptations)

Activation and deactivation of contexts

- Context can be in 2 states:
  - active *or*
  - inactive
- Context becomes active after an **activation** request
- Context becomes inactive after a **deactivation** request

$\Rightarrow$ That's nice, but how to *model* it?

Activation and deactivation of contexts

- Context can be in 2 states:
  - active *or*
  - inactive
- Context becomes active after an **activation** request
- Context becomes inactive after a **deactivation** request

$\Rightarrow$ That's nice, but how to *model* it?

Plan

1. Introduction

2. Context-oriented programming

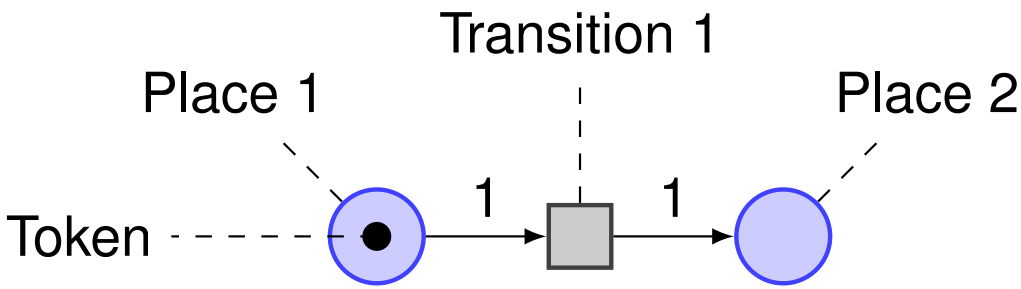
3. Context modelling

4. Conclusion

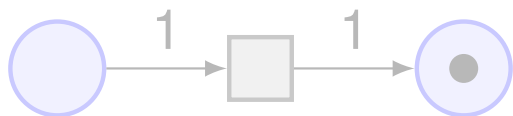
Petri Net

A Petri Net is a graph with **transitions** and **places**.

- Transitions are events
- Places are conditions



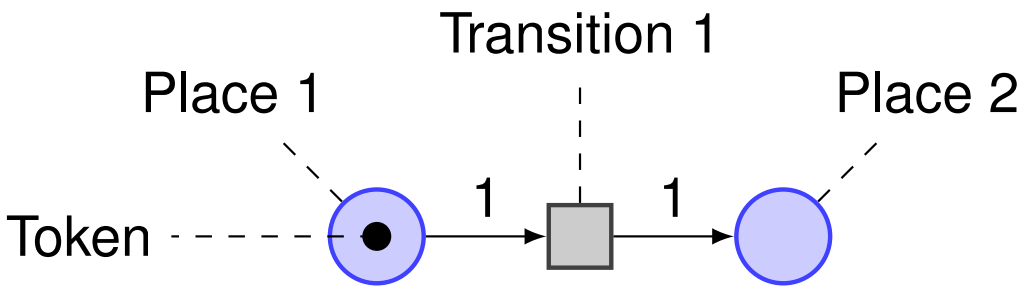
A transition can be **fired** if there are sufficient tokens in its input places.



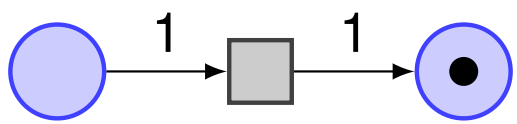
Petri Net

A Petri Net is a graph with **transitions** and **places**.

- Transitions are events
- Places are conditions



A transition can be **fired** if there are sufficient tokens in its input places.



Why Petri Nets?

- Petri Nets are good at modelling:
  - Choices
  - Iteration
  - Concurrent execution
  - Multiple activations (more than one token in a place)
- They have nice properties, such as:
  - Coverability
  - Reachability
  - Liveness

$\Rightarrow$ Those properties can be used to **identify conflicts**!

Why Petri Nets?

- Petri Nets are good at modelling:
  - Choices
  - Iteration
  - Concurrent execution
  - Multiple activations (more than one token in a place)
- They have nice properties, such as:
  - Coverability
  - Reachability
  - Liveness

$\Rightarrow$ Those properties can be used to **identify conflicts**!

Petri Net limitation

Property

COP systems must ensure the **reactivity** to changes in the surrounding execution environment.

Reactive systems have *perfect-synchrony hypothesis*:

- Outputs are produced instantaneously after the inputs occur.

Bad news

Petri Nets are not expressive enough for this hypothesis. . .

Petri Net limitation

Property

COP systems must ensure the **reactivity** to changes in the surrounding execution environment.

Reactive systems have *perfect-synchrony hypothesis*:

- Outputs are produced instantaneously after the inputs occur.

Bad news

Petri Nets are not expressive enough for this hypothesis. . .

Reactive Petri Net

Reactive Petri Net splits transitions into

- **External** transitions □ that **may** fire
- **Internal** transitions ■ that **must** fire

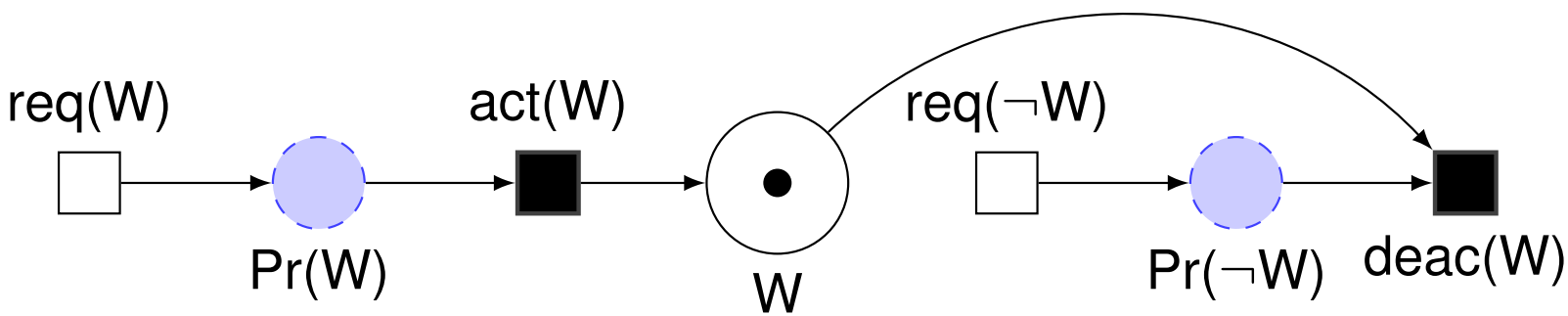


Figure : Example of a `WiFi` (W) context (de)activation Petri Net

Relation between contexts

A context-oriented system has **multiple** contexts.

Contexts can be related to each others.

Example A system with 3 contexts:

- WiFi context (W)
- 3G context (3G)
- Connectivity context (C)

If WiFi is enabled or 3G is enabled, the connectivity must be enabled too.

This is called a **disjunction dependency relation**.

Relation between contexts

A context-oriented system has **multiple** contexts.

Contexts can be related to each others.

Example A system with 3 contexts:

- WiFi context (W)
- 3G context (3G)
- Connectivity context (C)

If WiFi is enabled or 3G is enabled, the connectivity must be enabled too.

This is called a **disjunction dependency relation**.

Relation between contexts

A context-oriented system has **multiple** contexts.

Contexts can be related to each others.

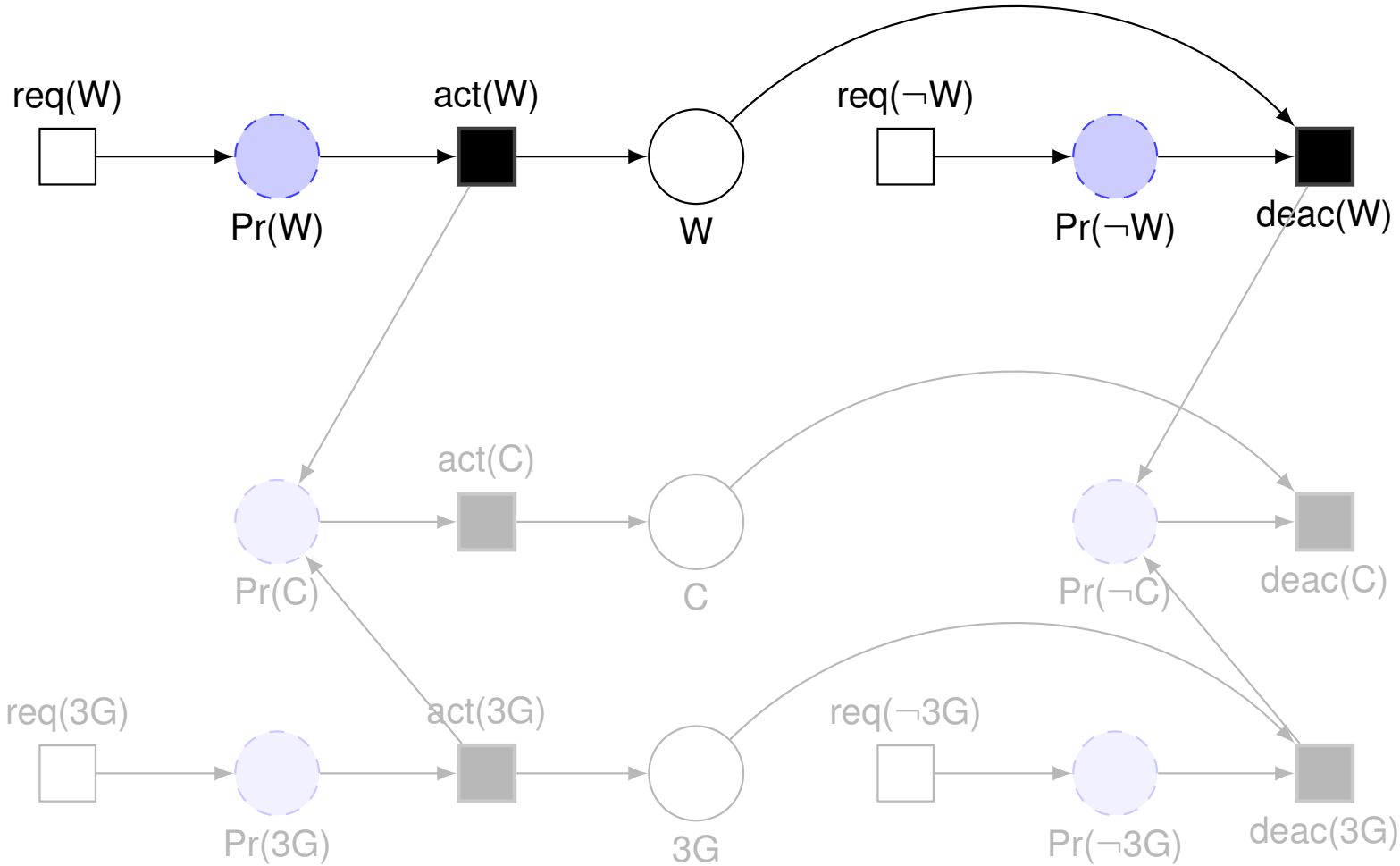
Example A system with 3 contexts:

- WiFi context (W)
- 3G context (3G)
- Connectivity context (C)

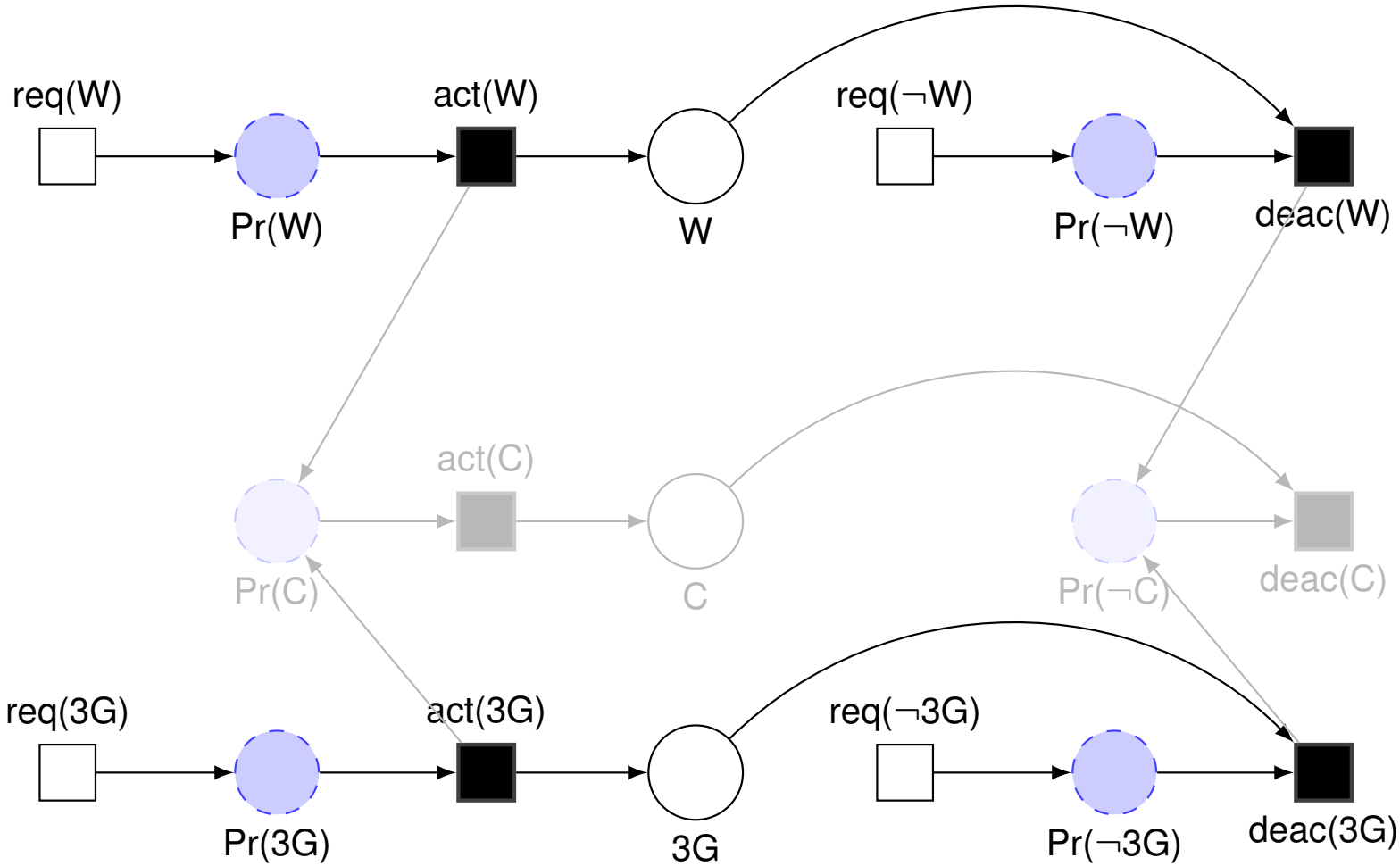
If WiFi is enabled or 3G is enabled, the connectivity must be enabled too.

This is called a **disjunction dependency relation**.

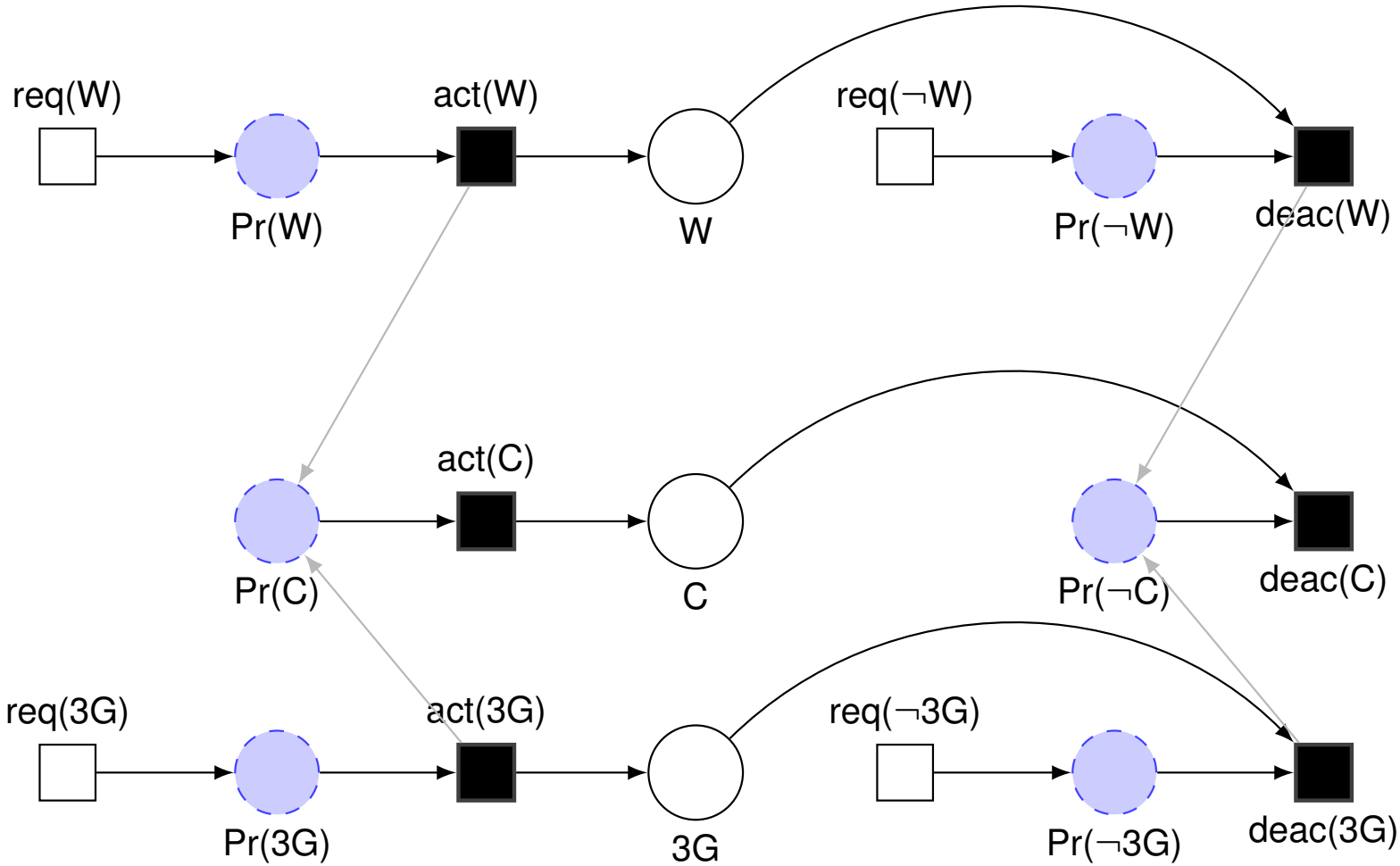
Relation between contexts



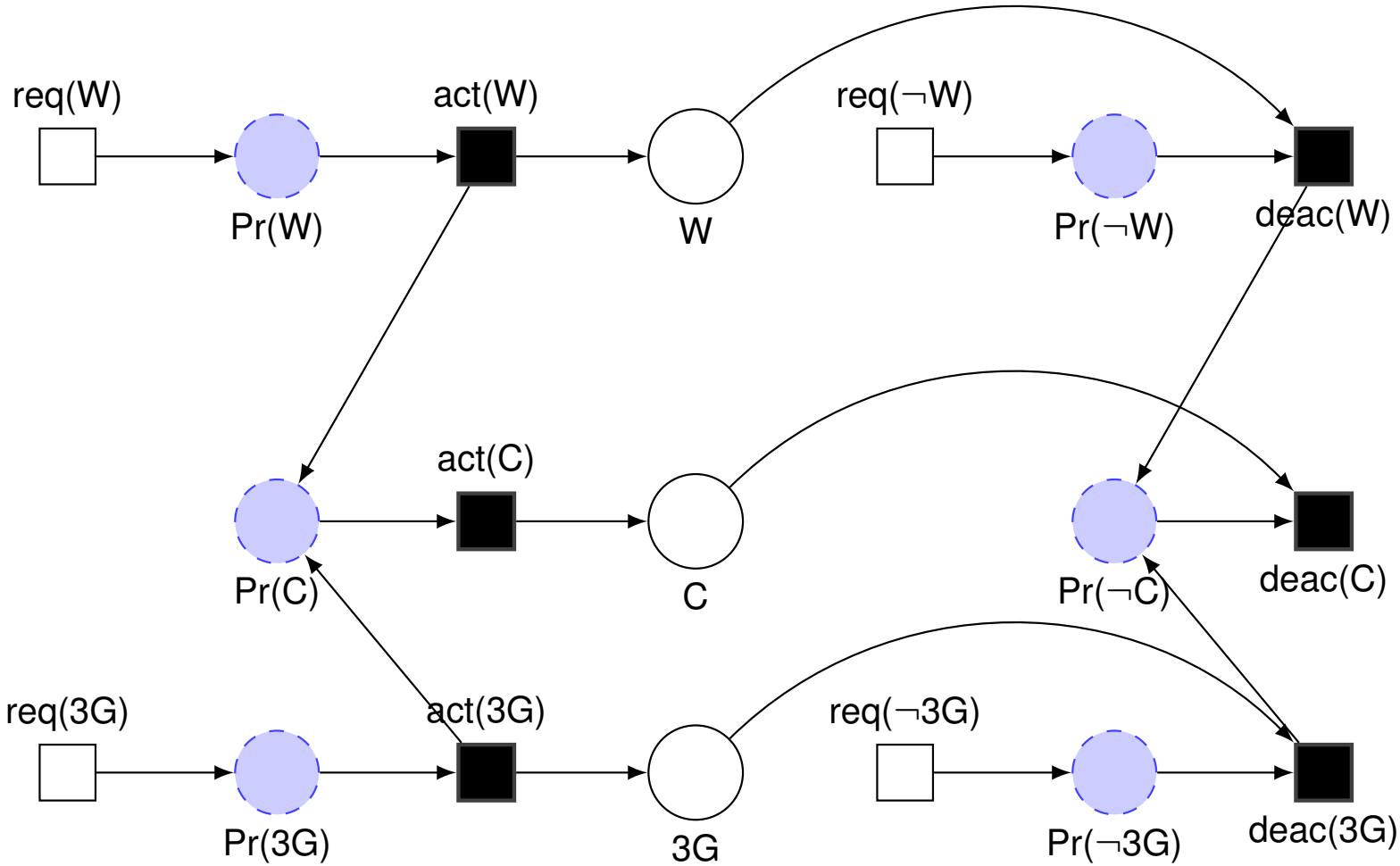
Relation between contexts



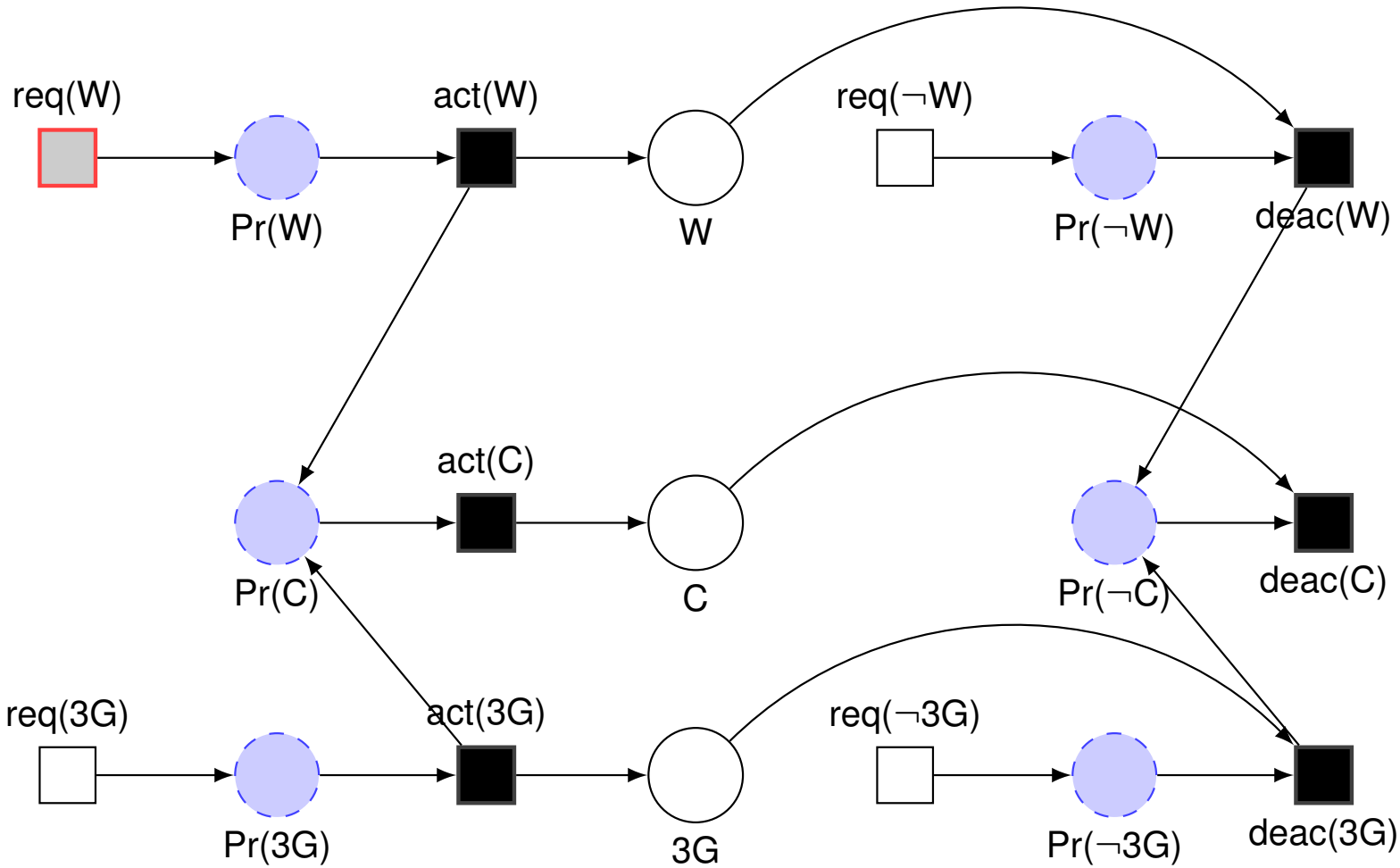
Relation between contexts



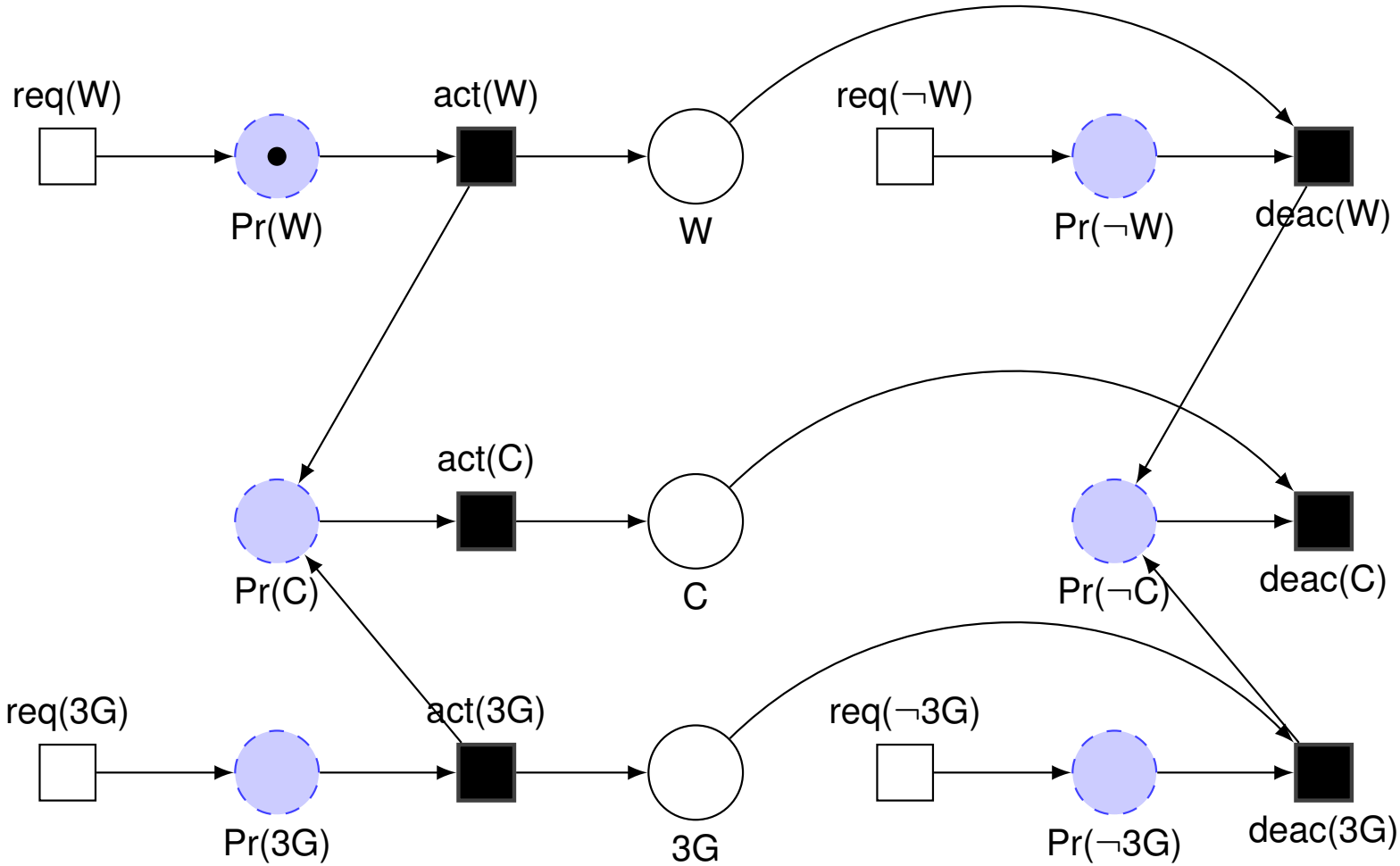
Relation between contexts



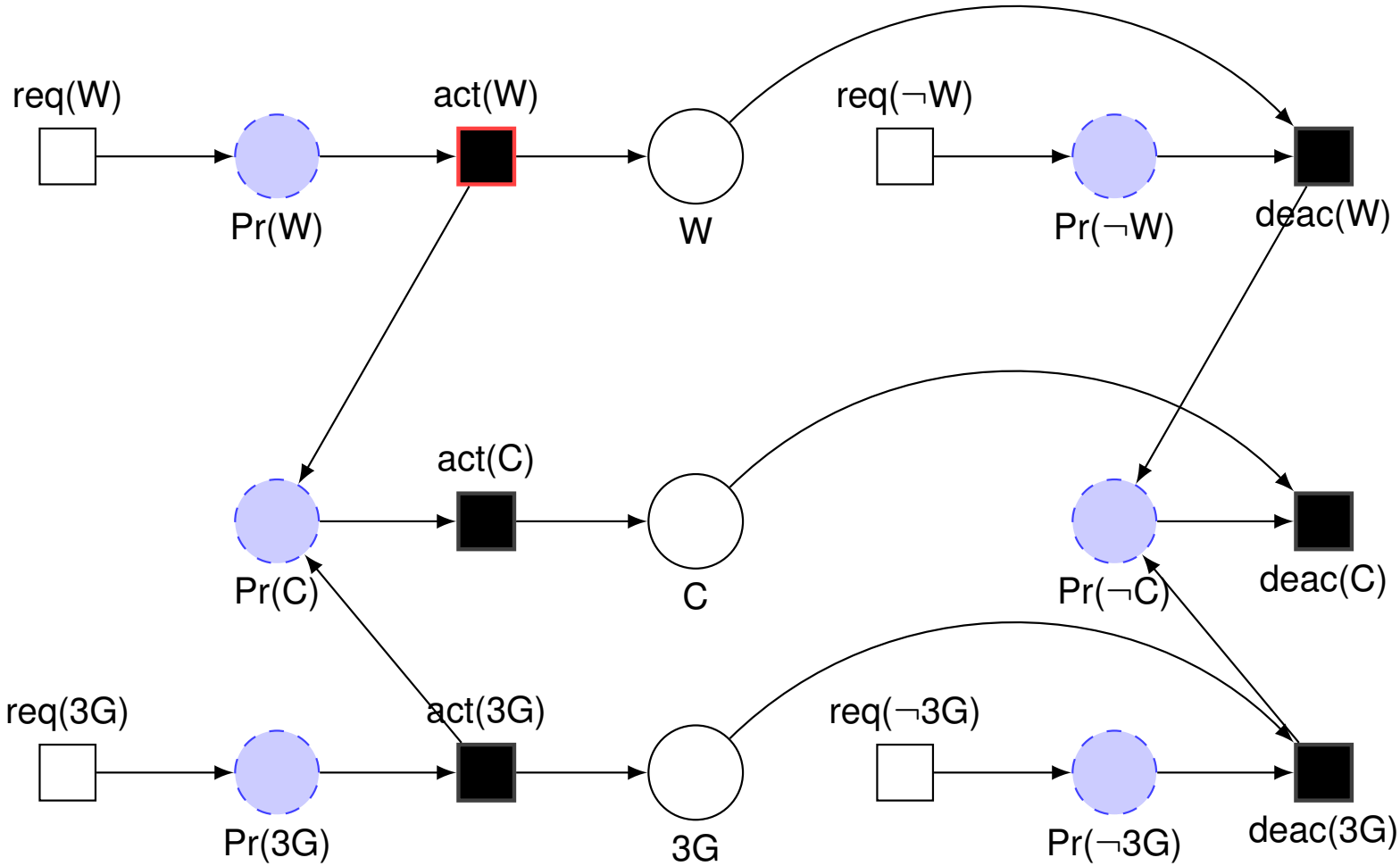
Relation between contexts



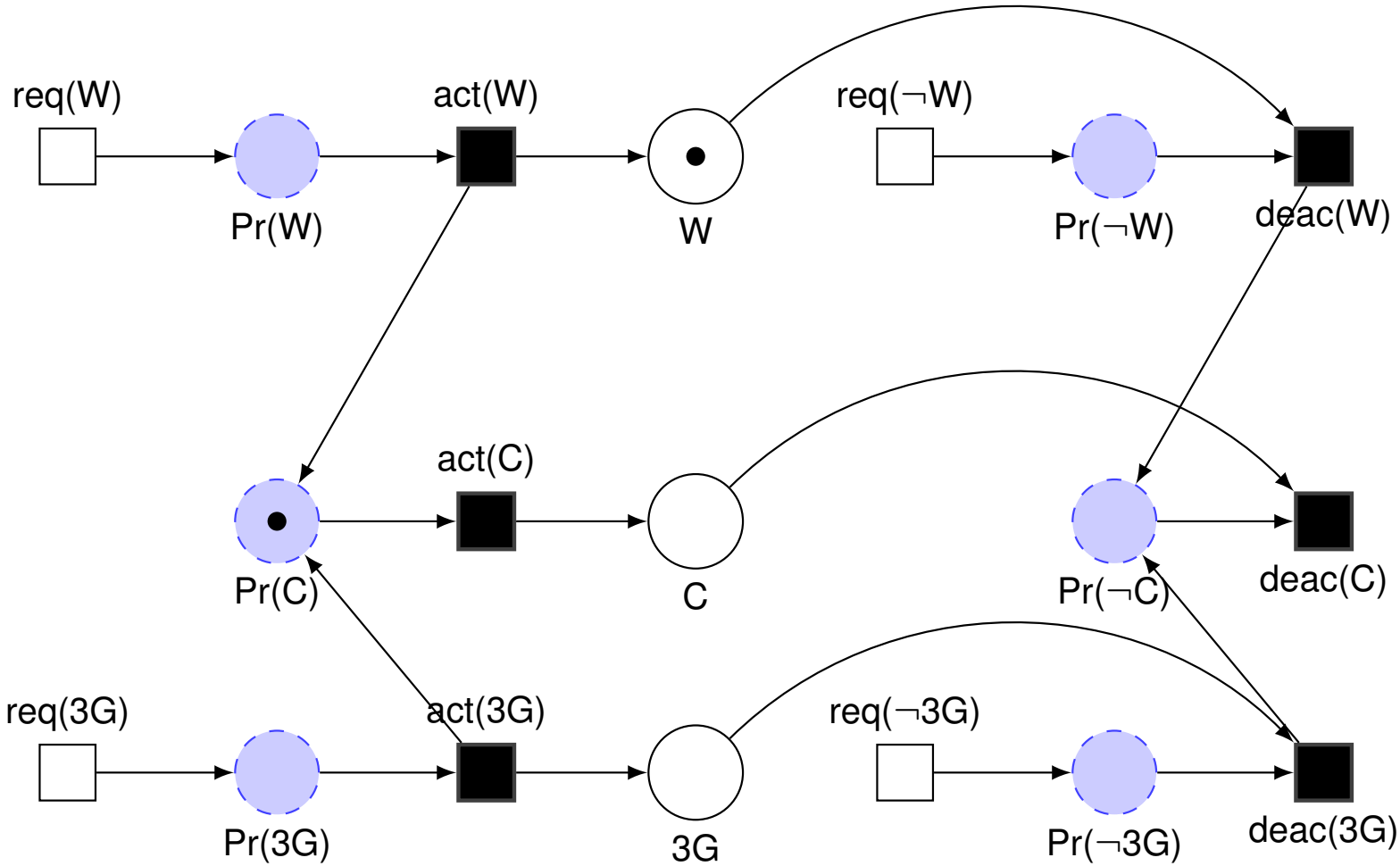
Relation between contexts



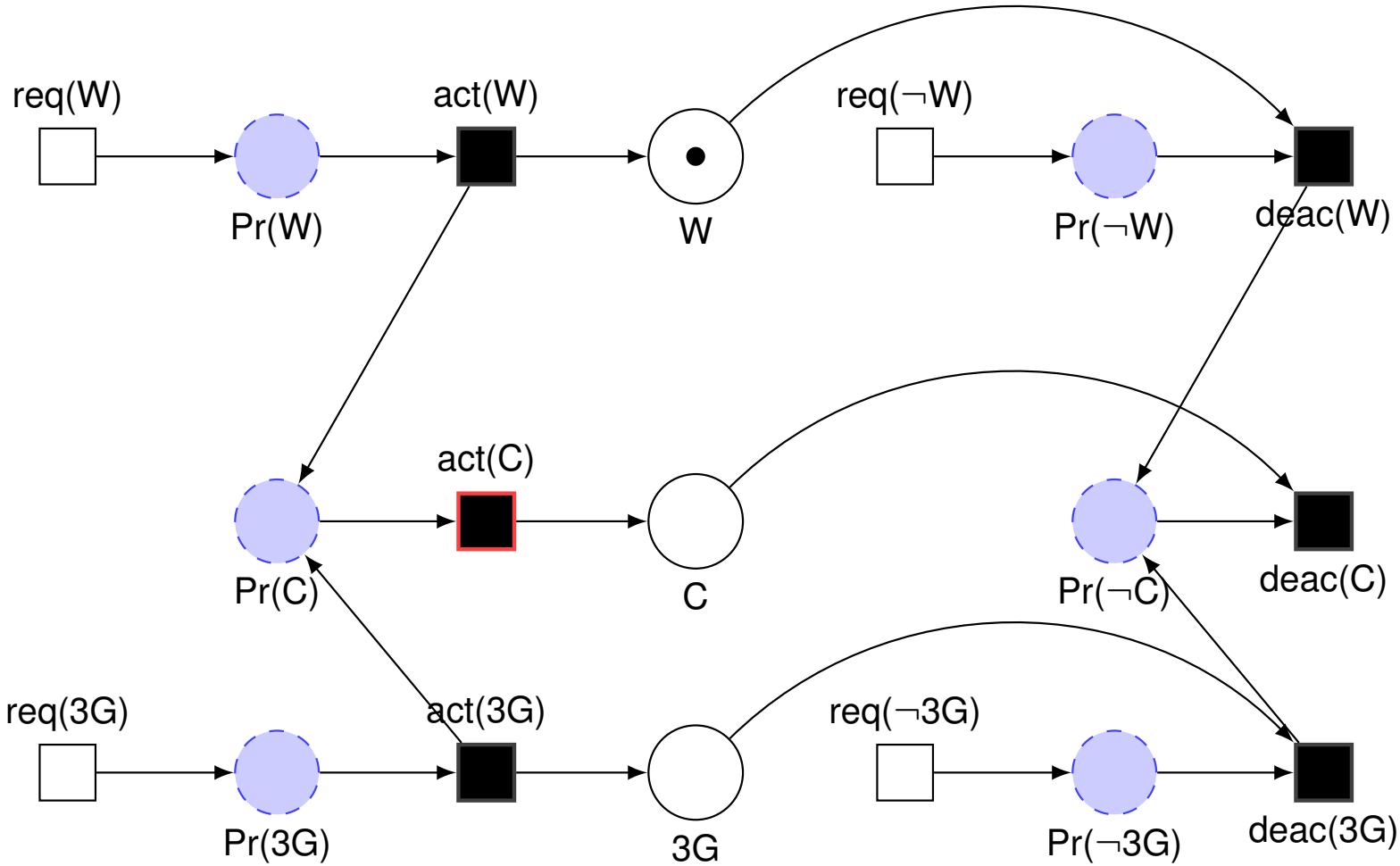
Relation between contexts



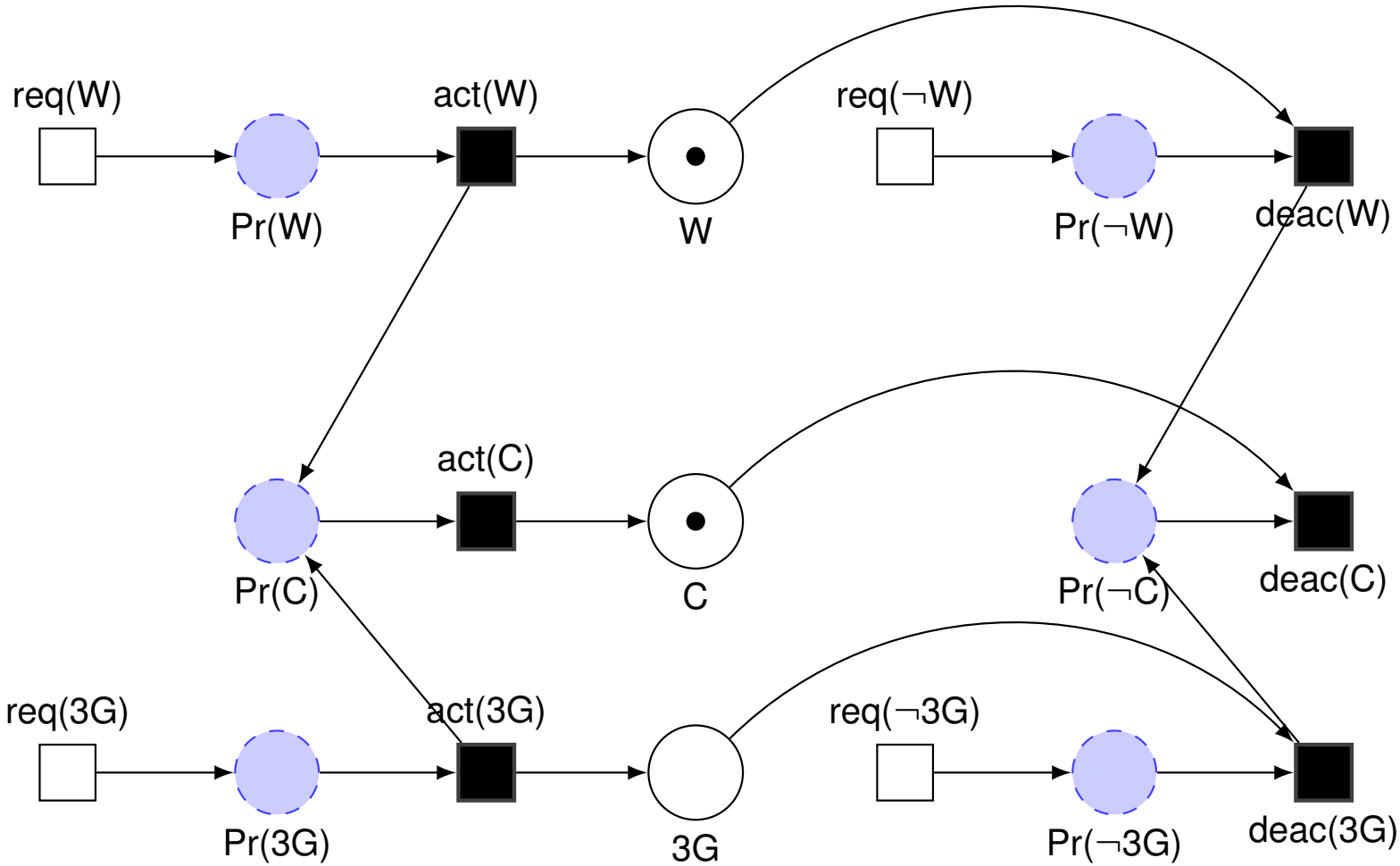
Relation between contexts



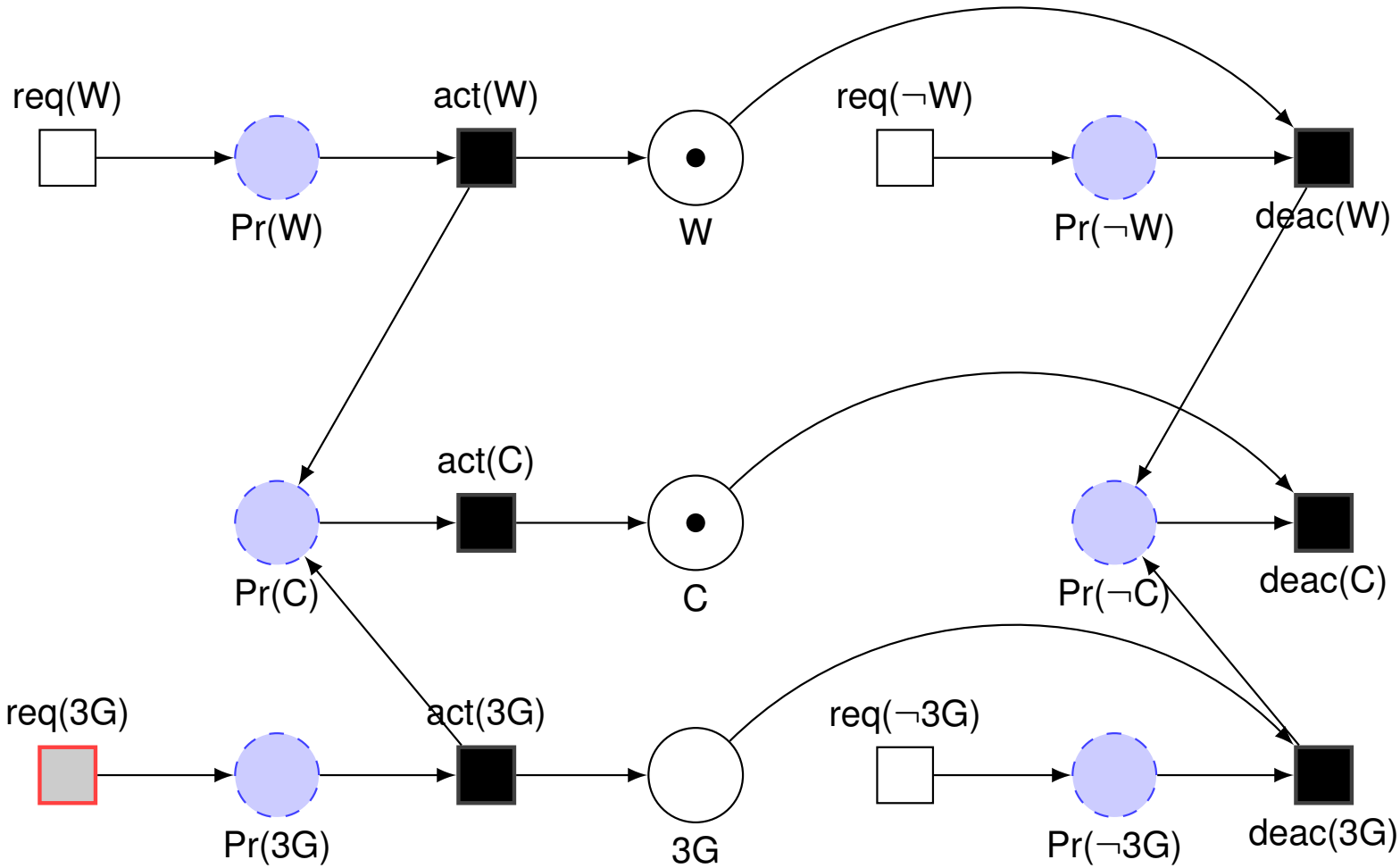
Relation between contexts



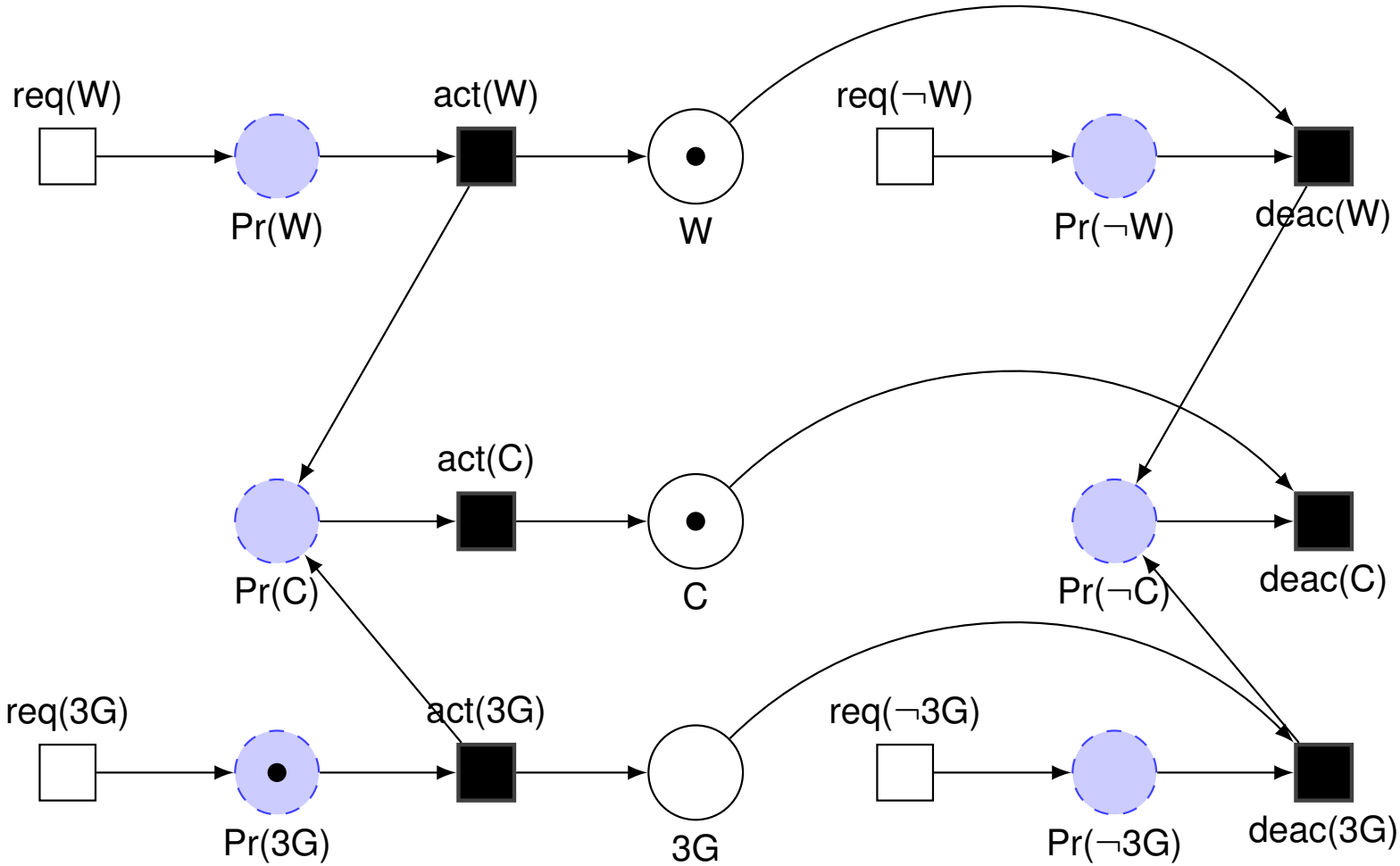
Relation between contexts



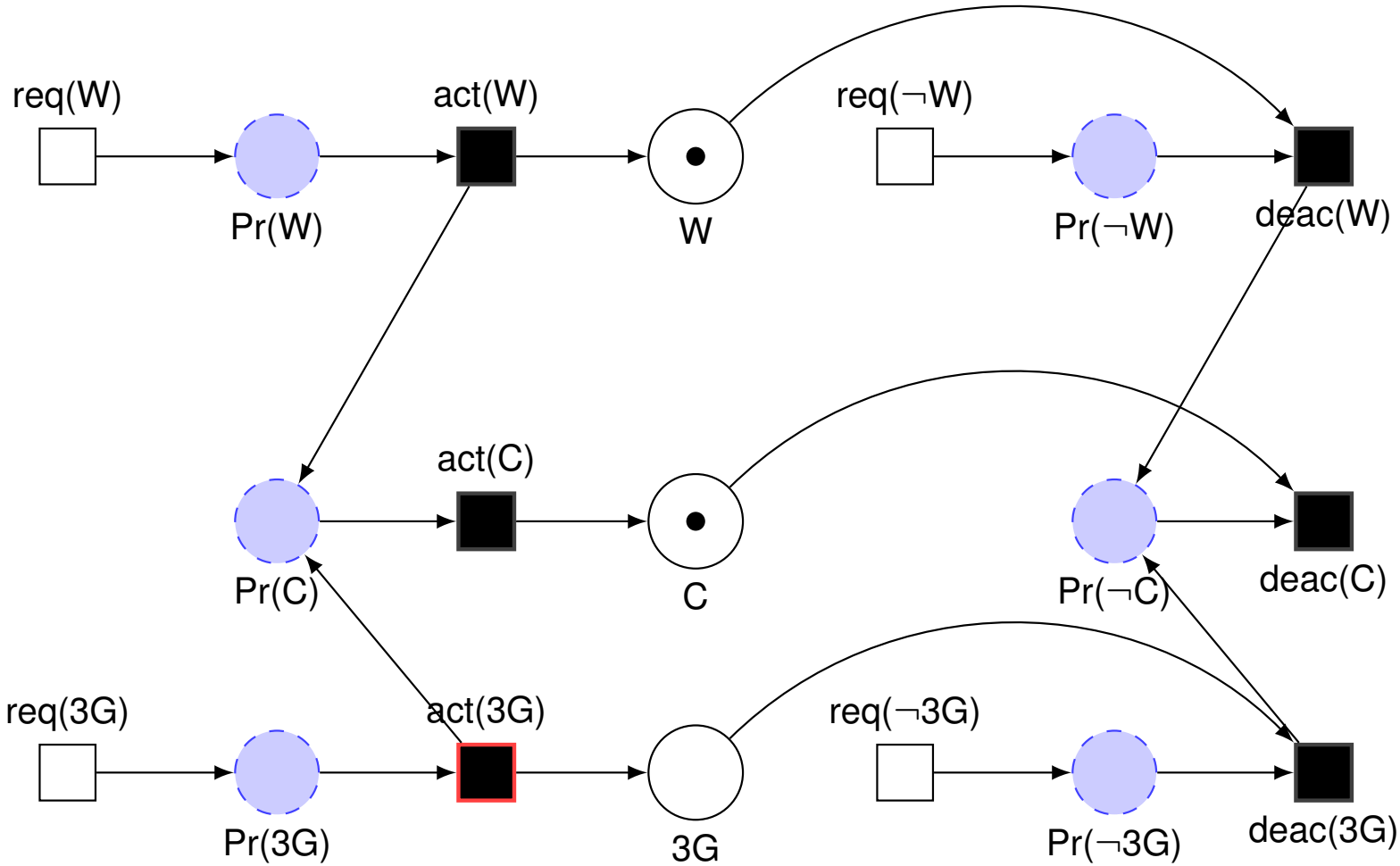
Relation between contexts



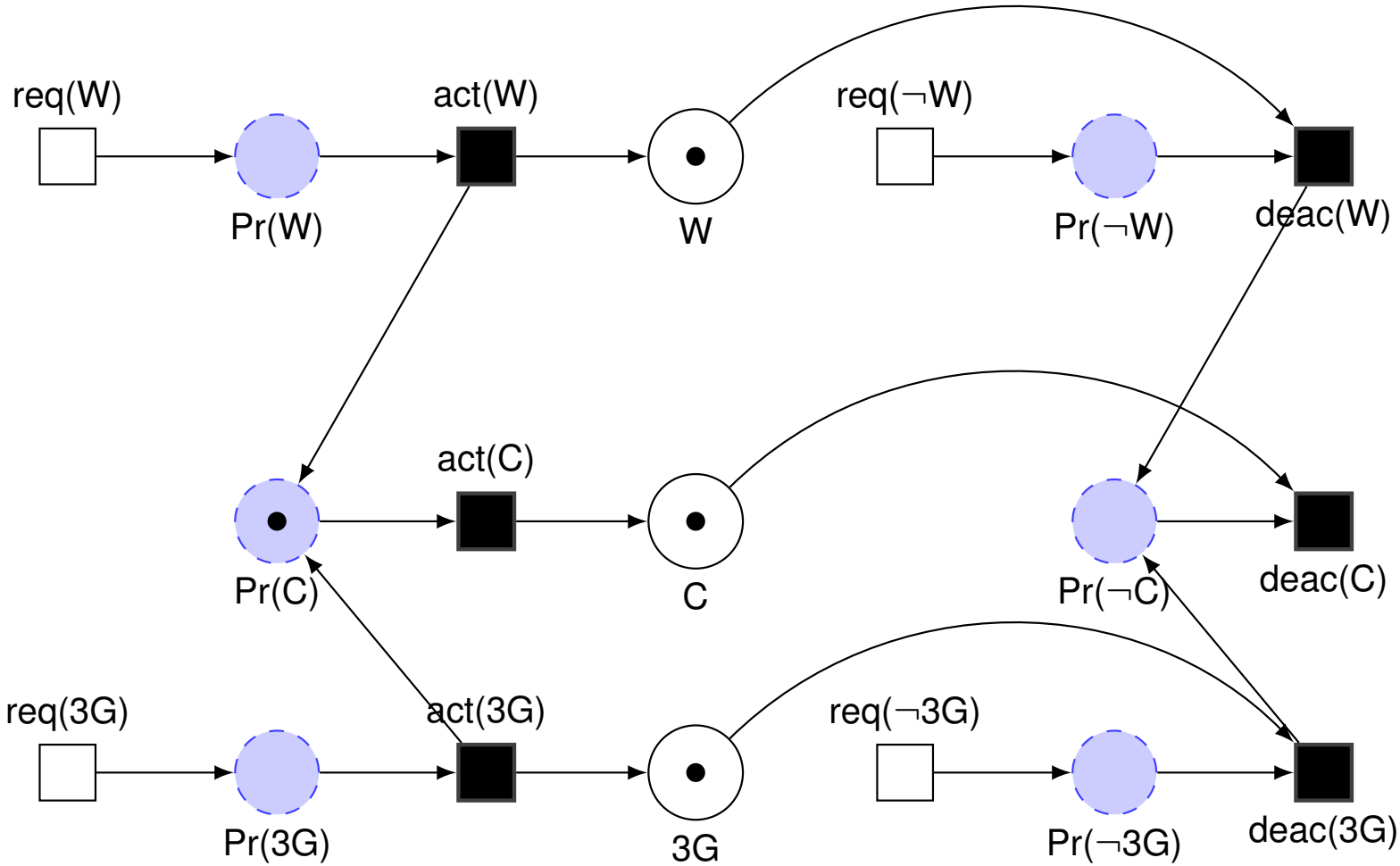
Relation between contexts



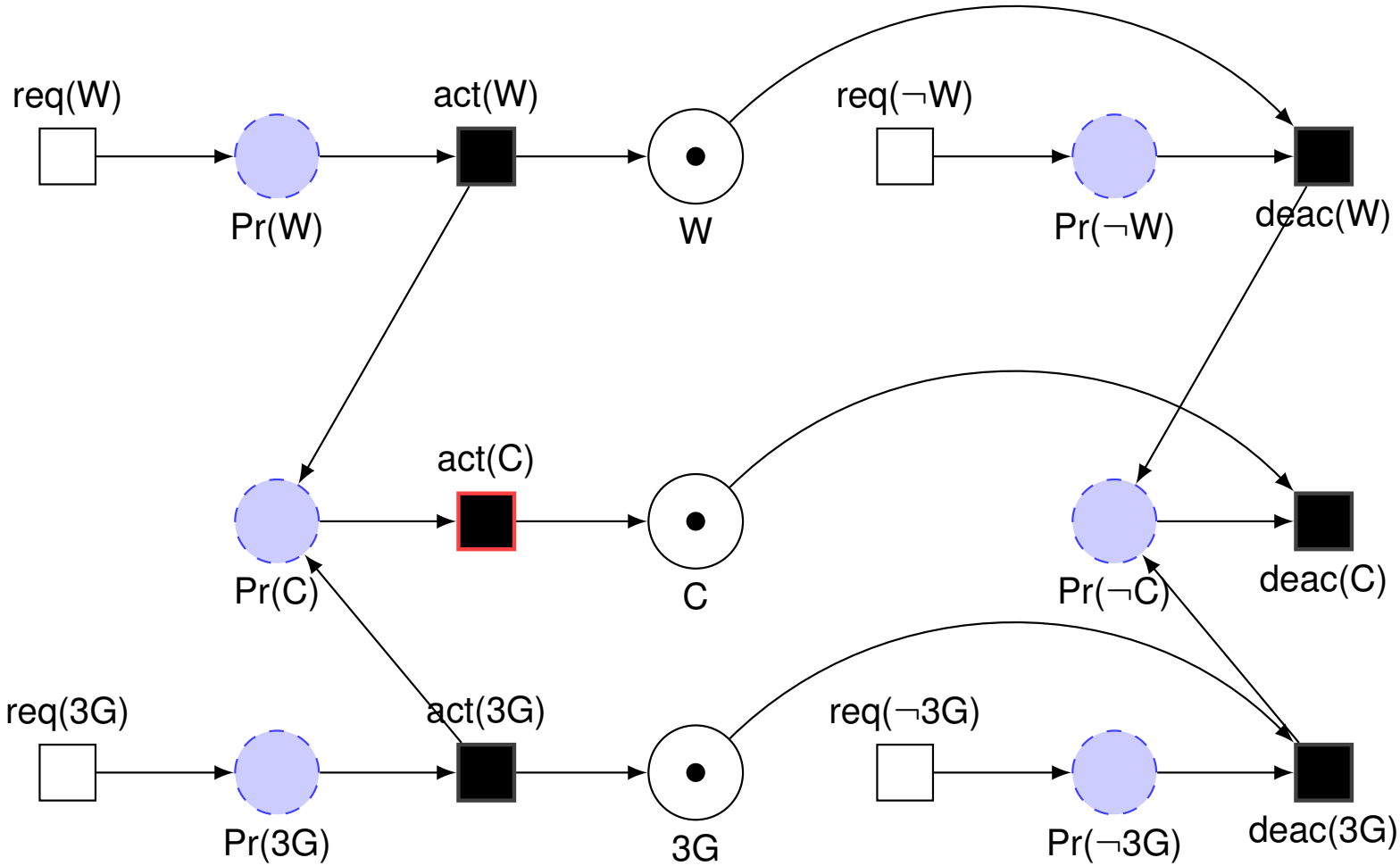
Relation between contexts



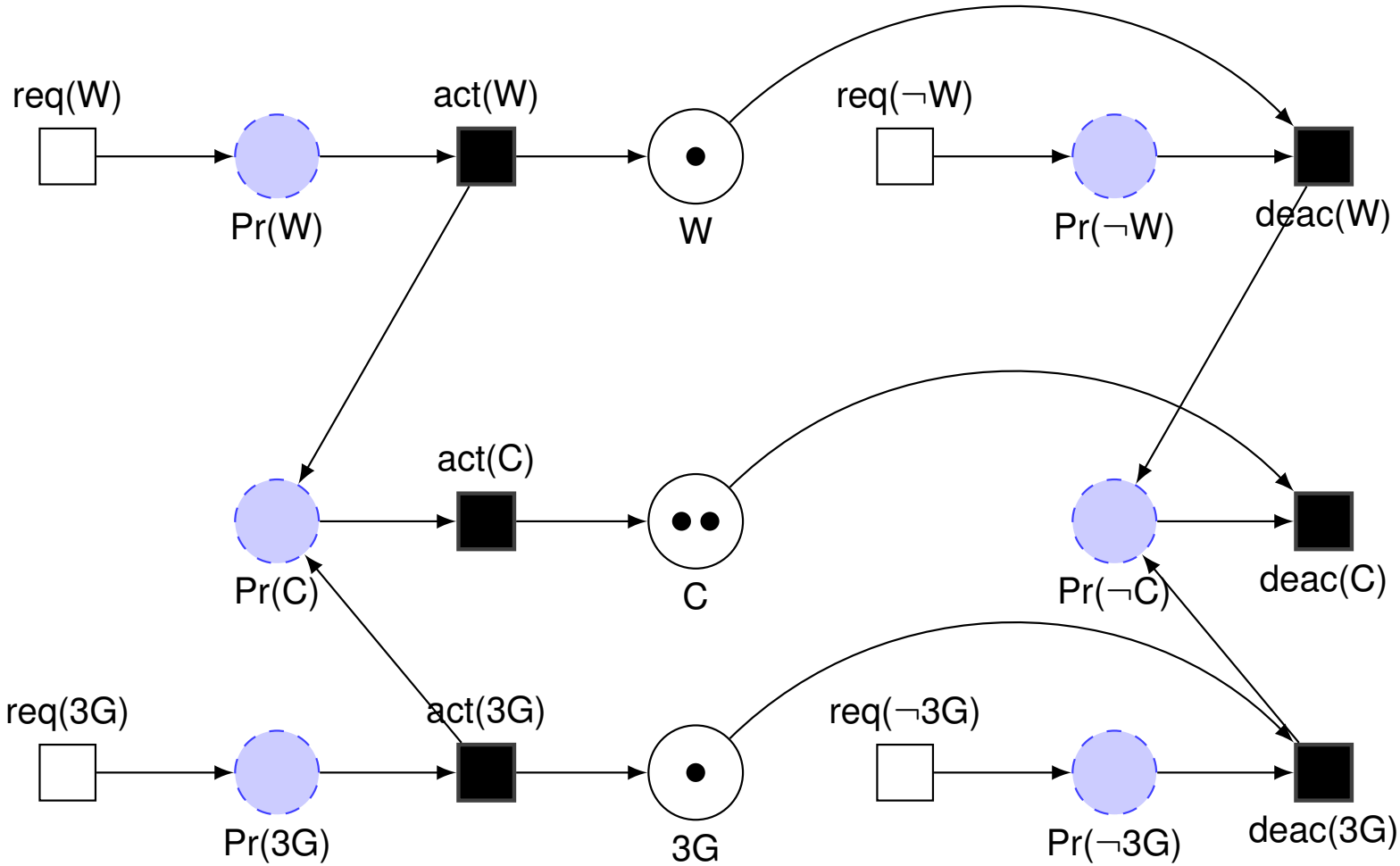
Relation between contexts



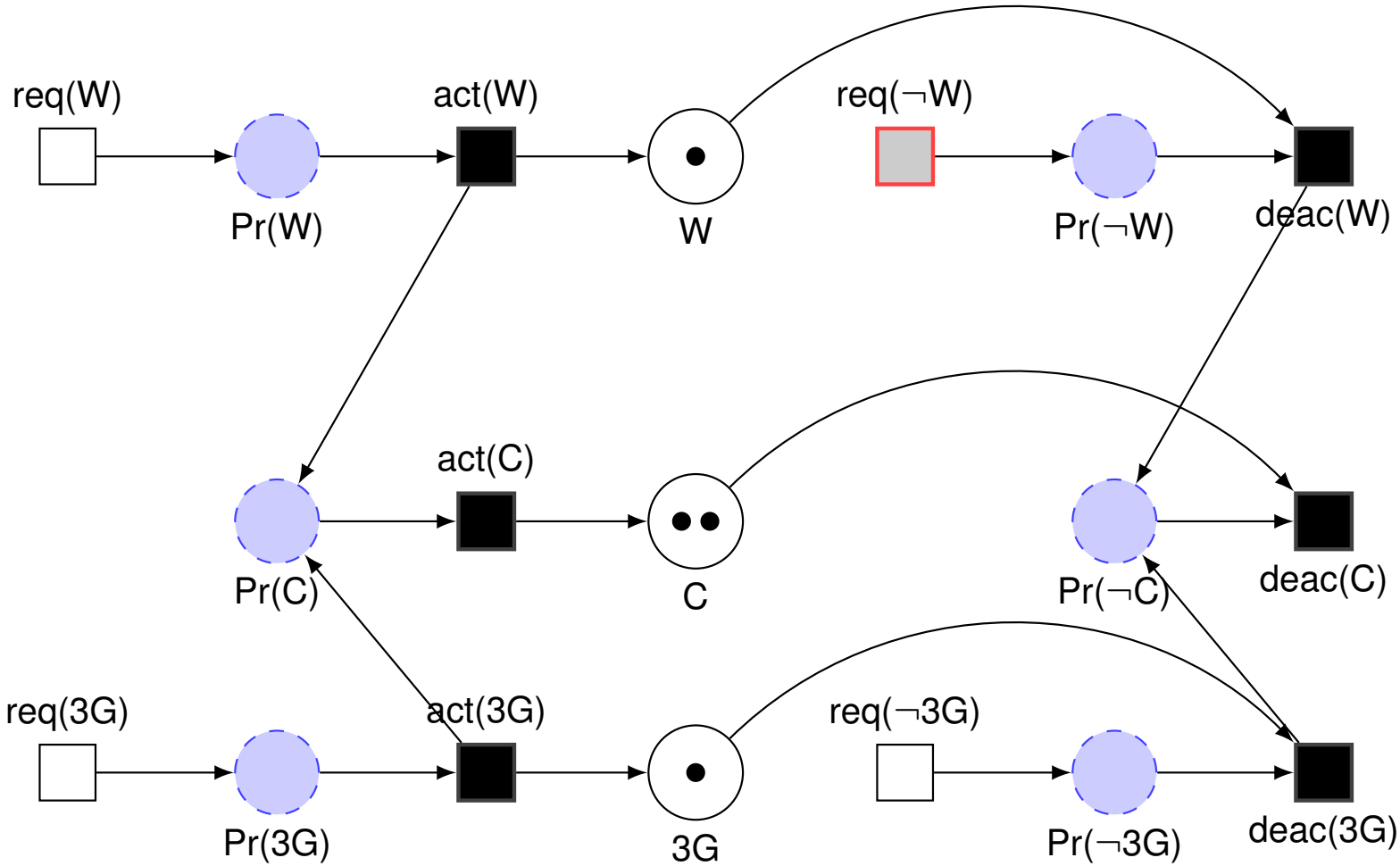
Relation between contexts



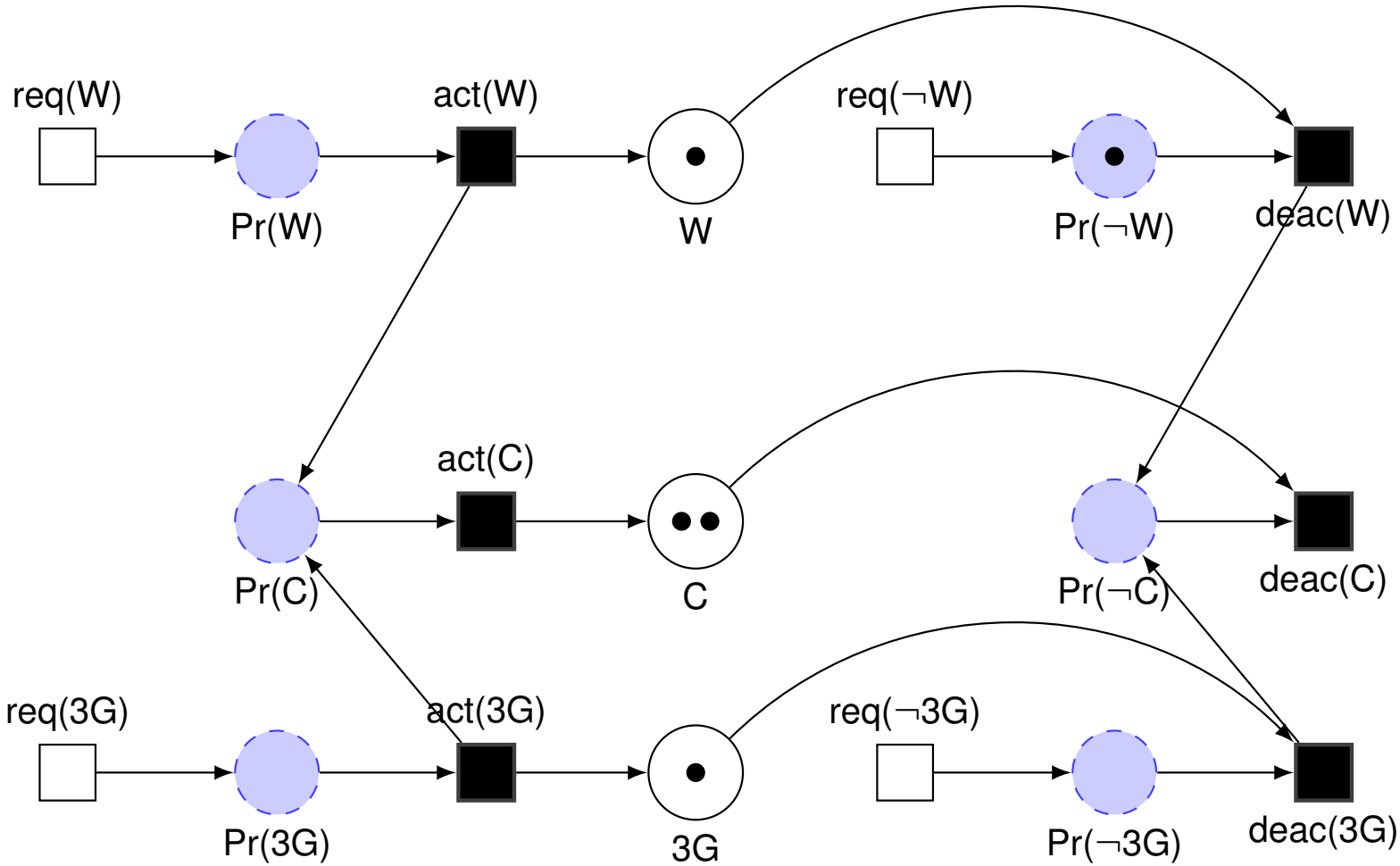
Relation between contexts



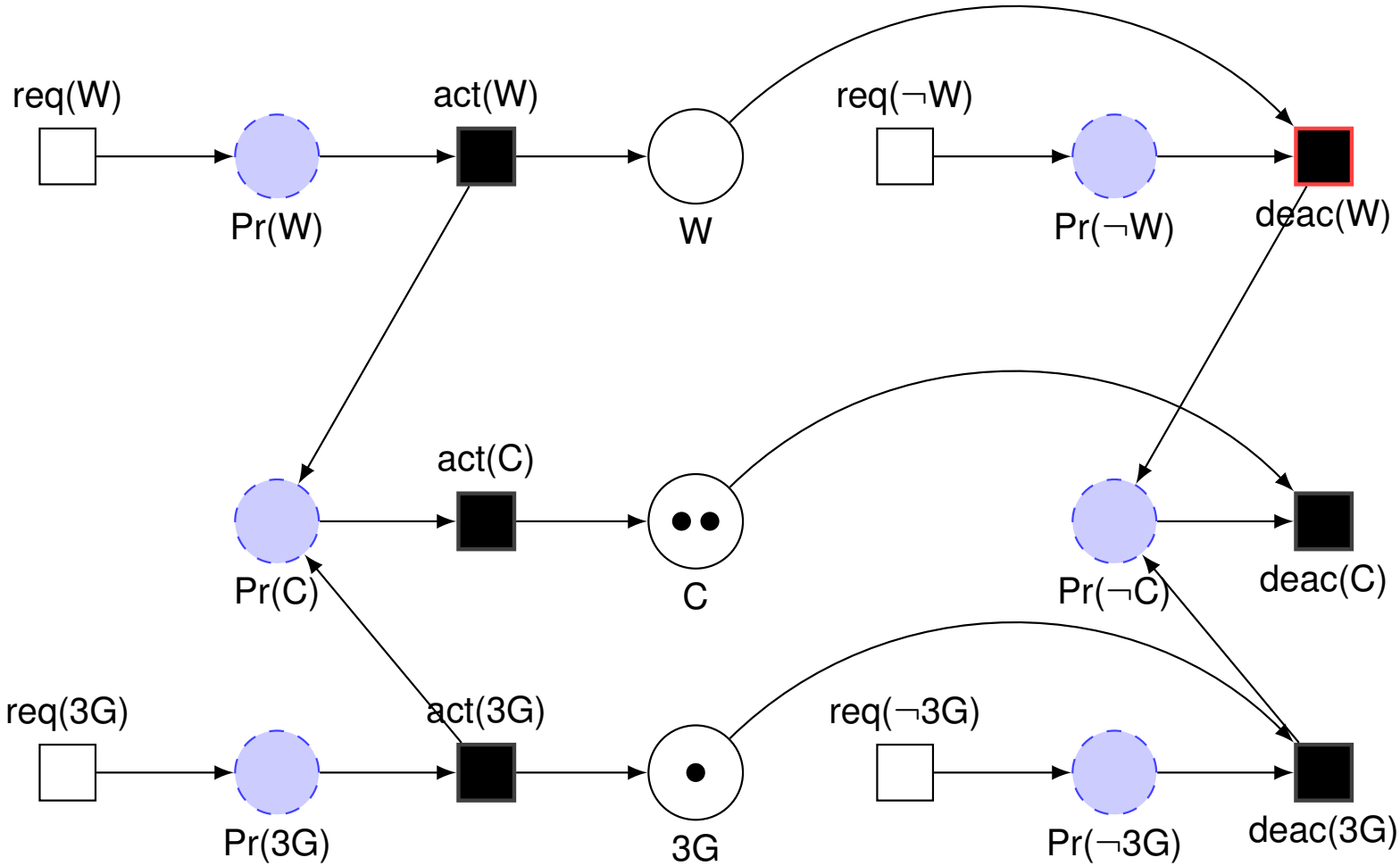
Relation between contexts



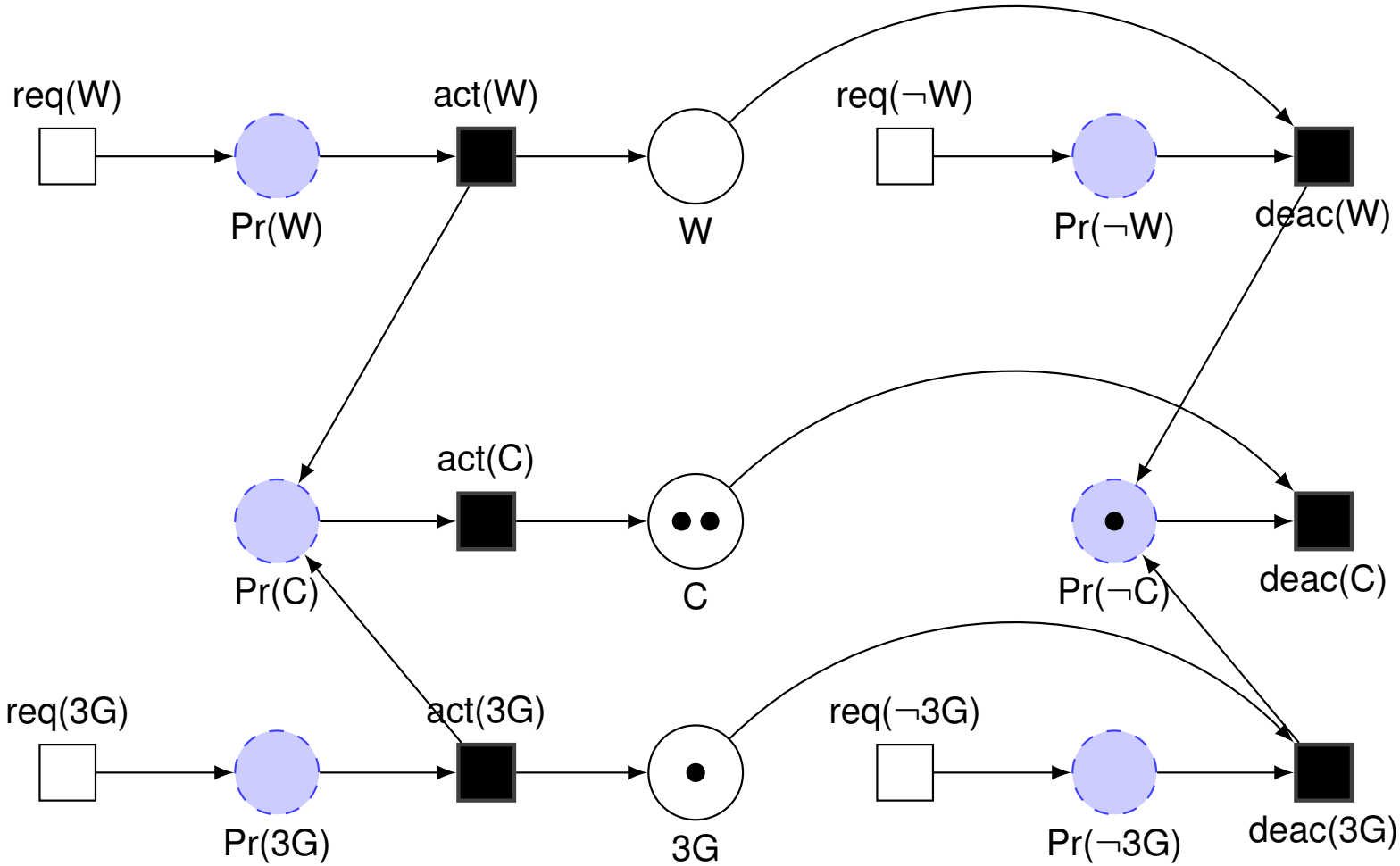
Relation between contexts



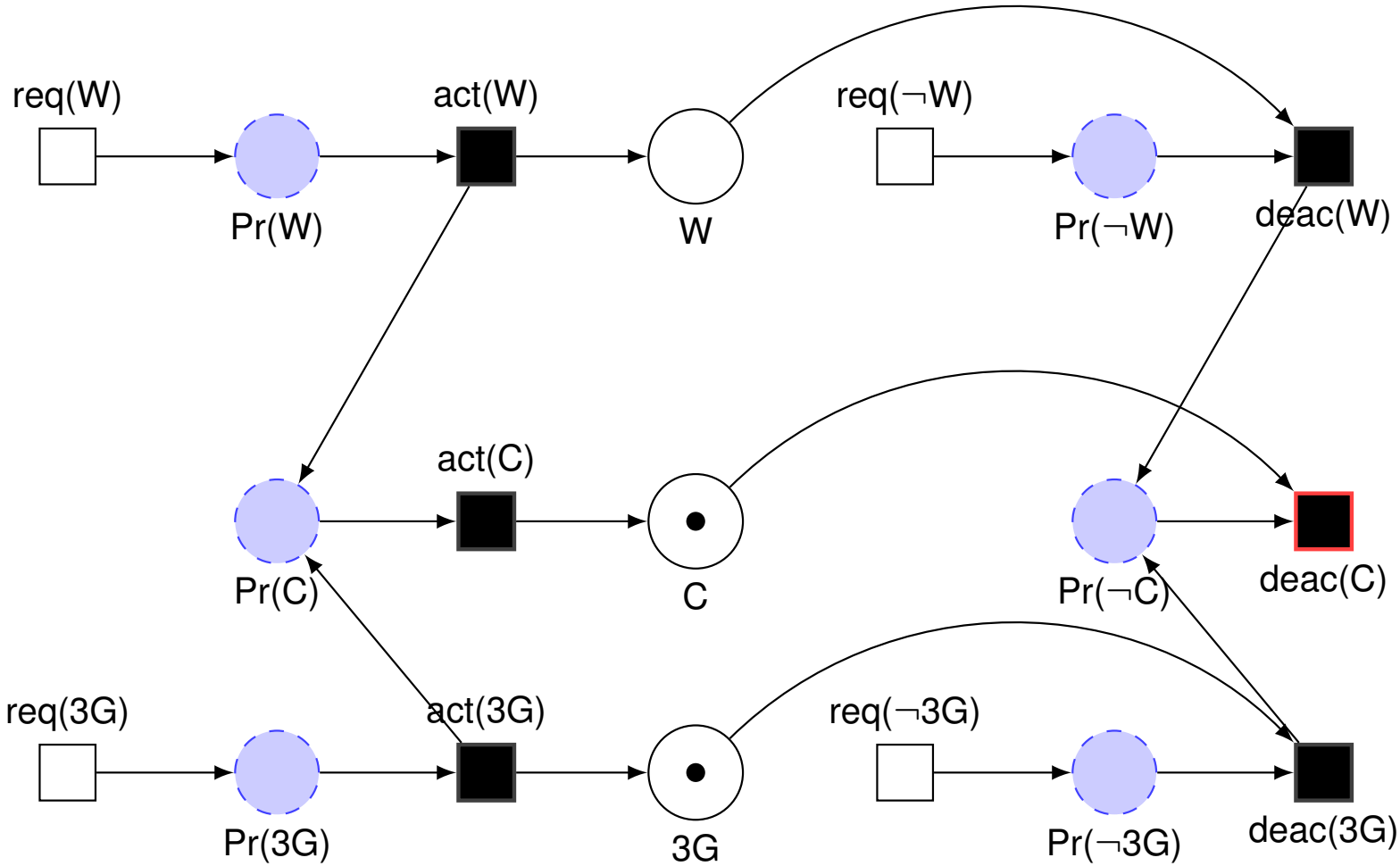
Relation between contexts



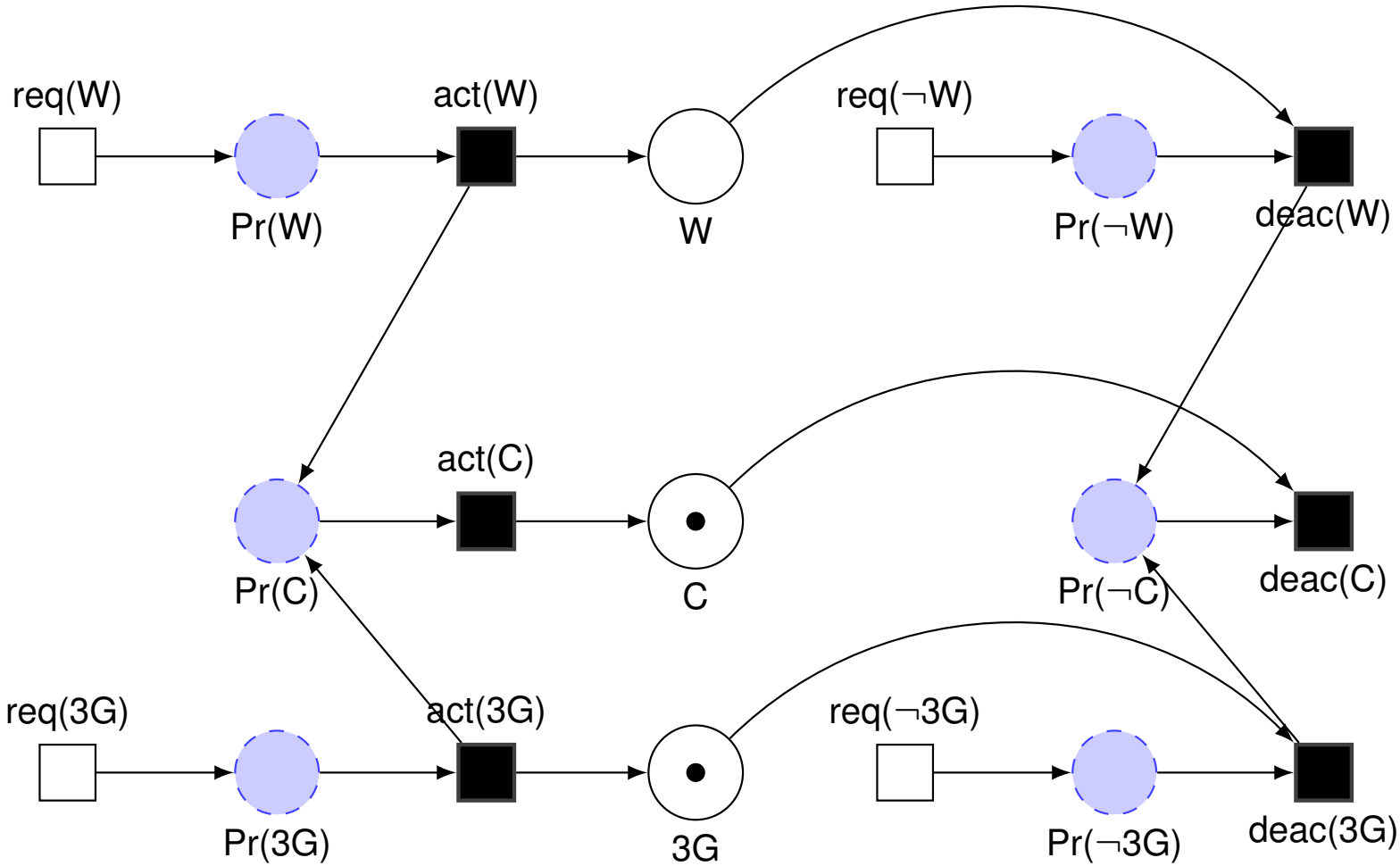
Relation between contexts



Relation between contexts



Relation between contexts



Causality between contexts

Bad news

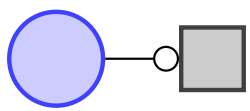
The relations between contexts are sometimes more complex. . .

Connectivity activates an **AudioStream** context which will be deactivated as soon as **Connectivity** becomes inactive.

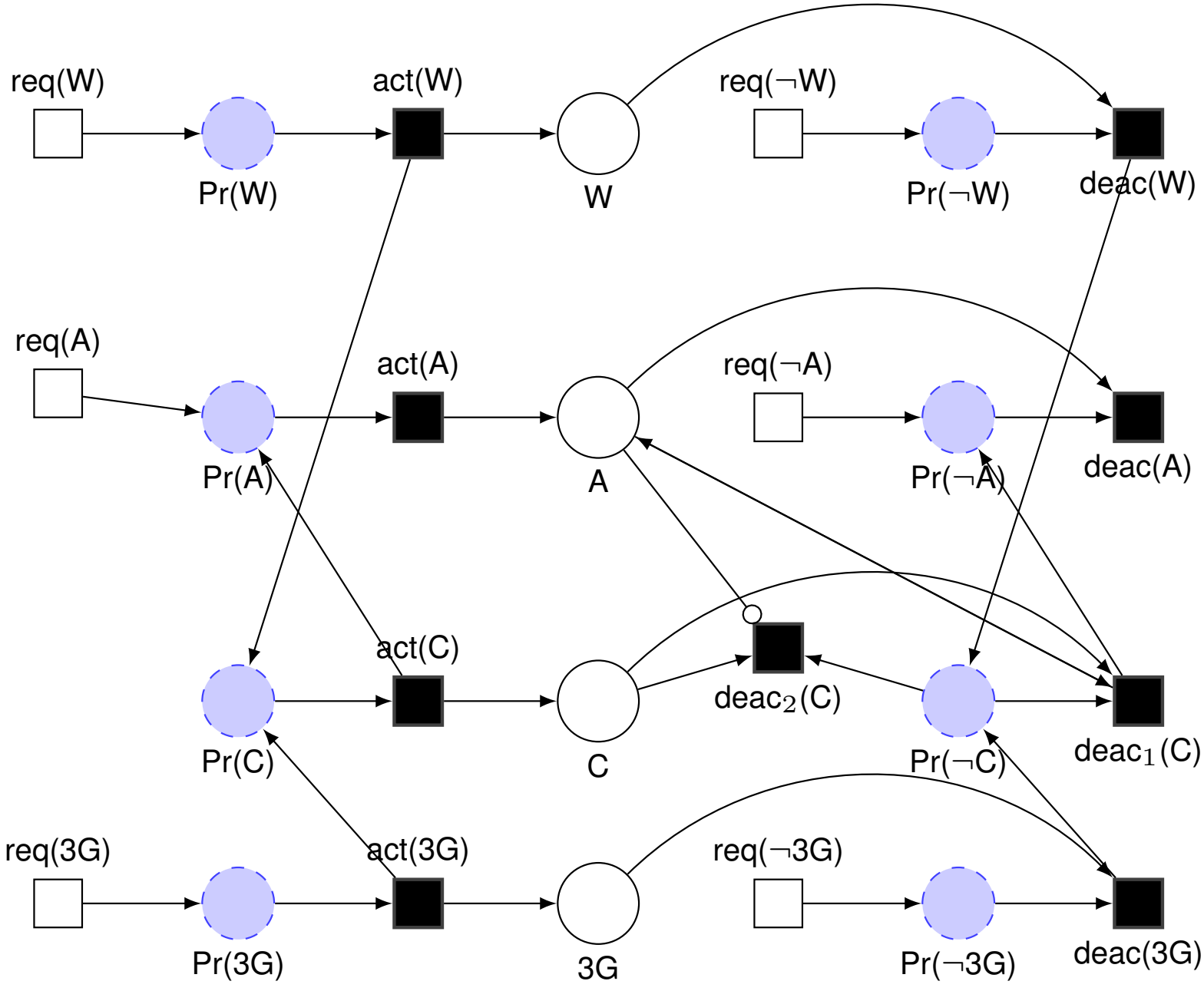
Relation type

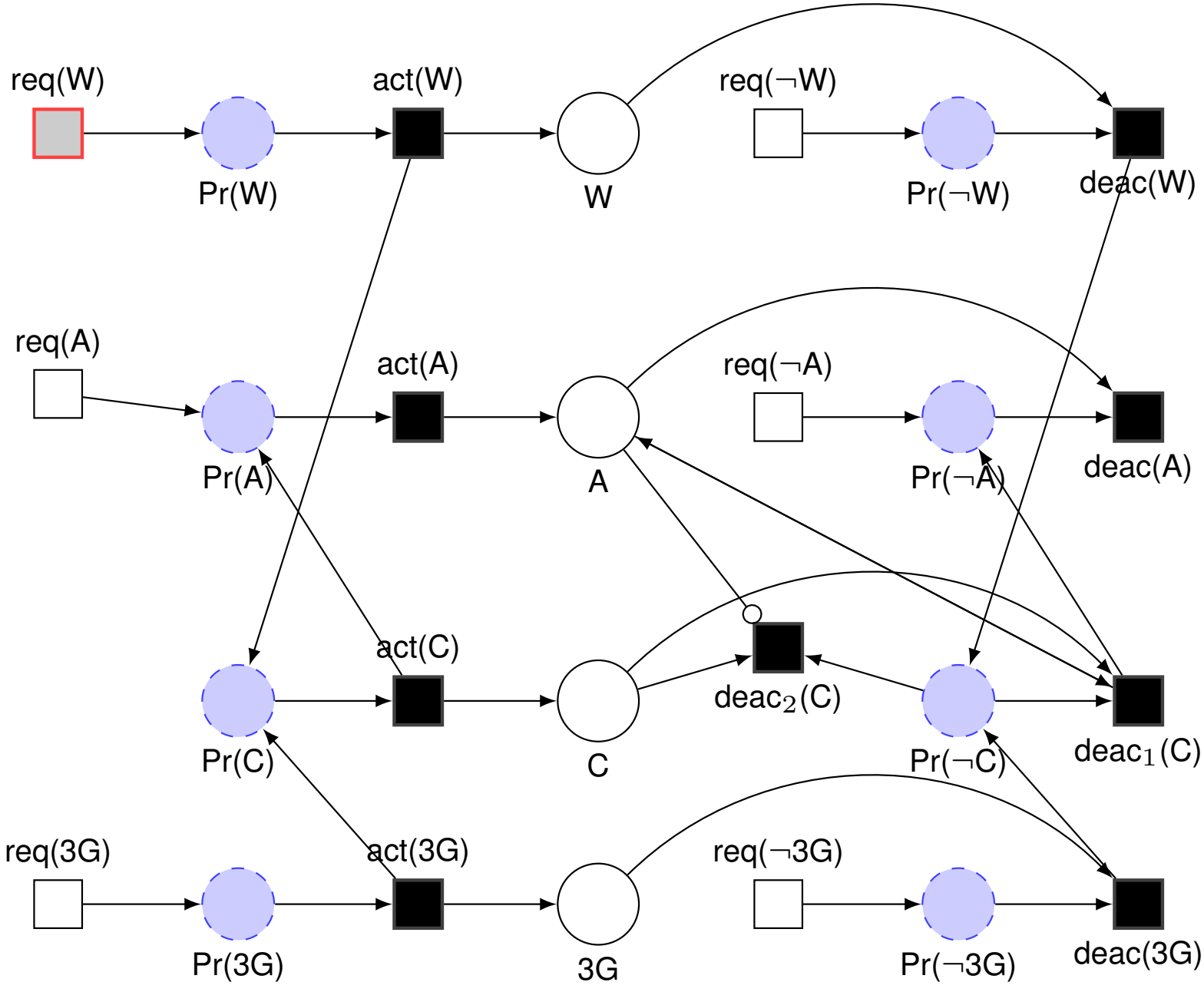
This is called a *causality dependency relation*.

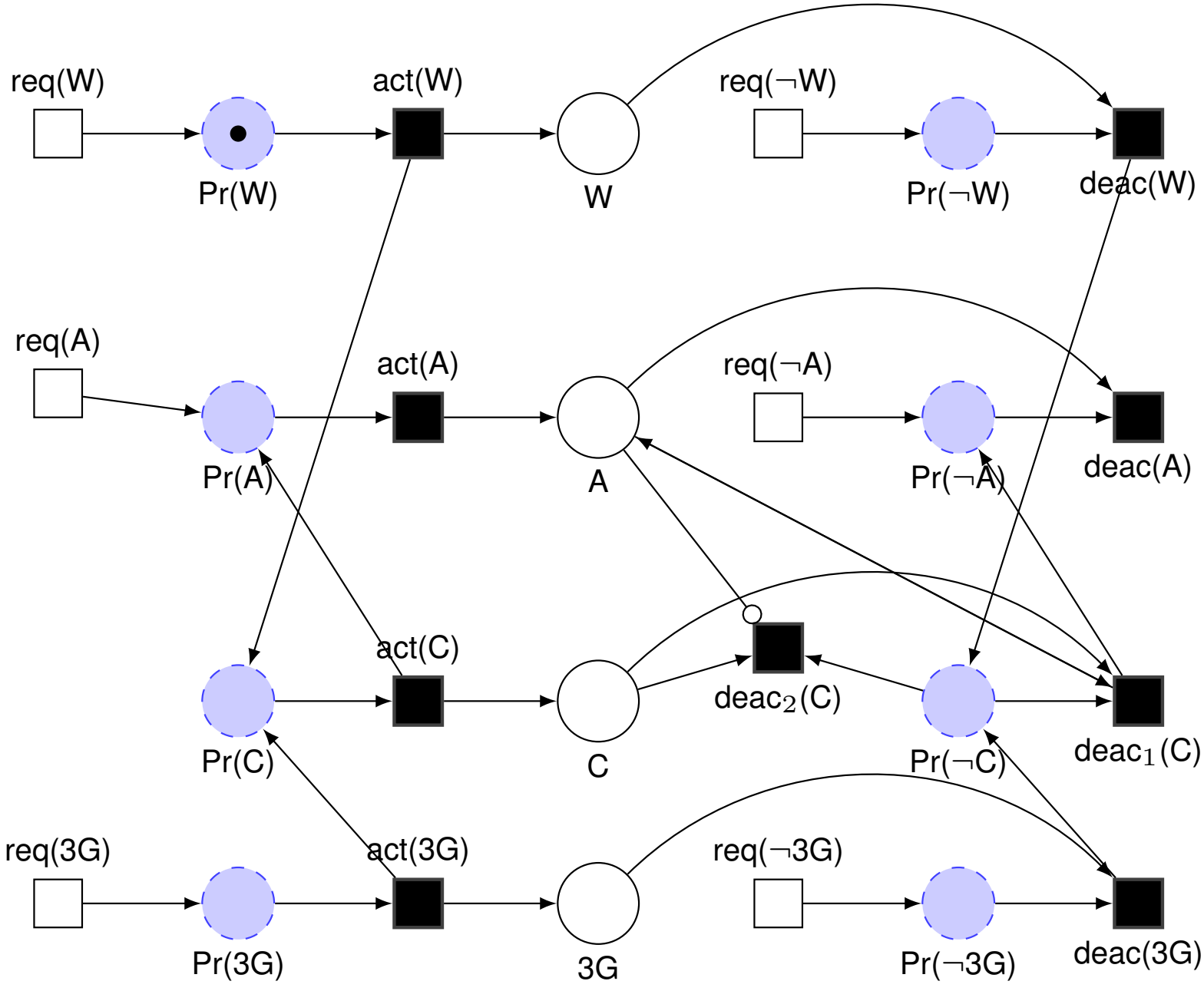
We add a constraint on transitions:

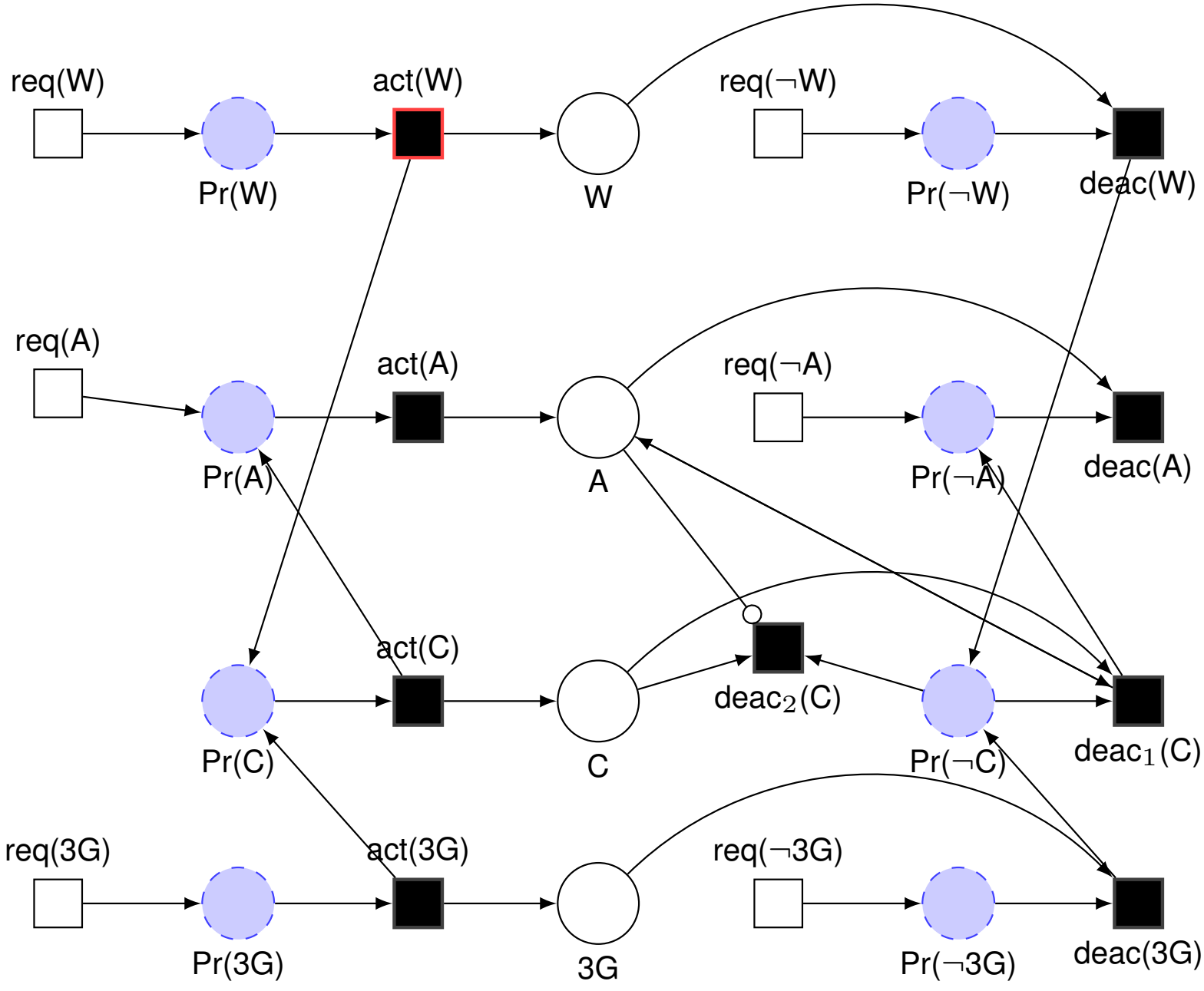


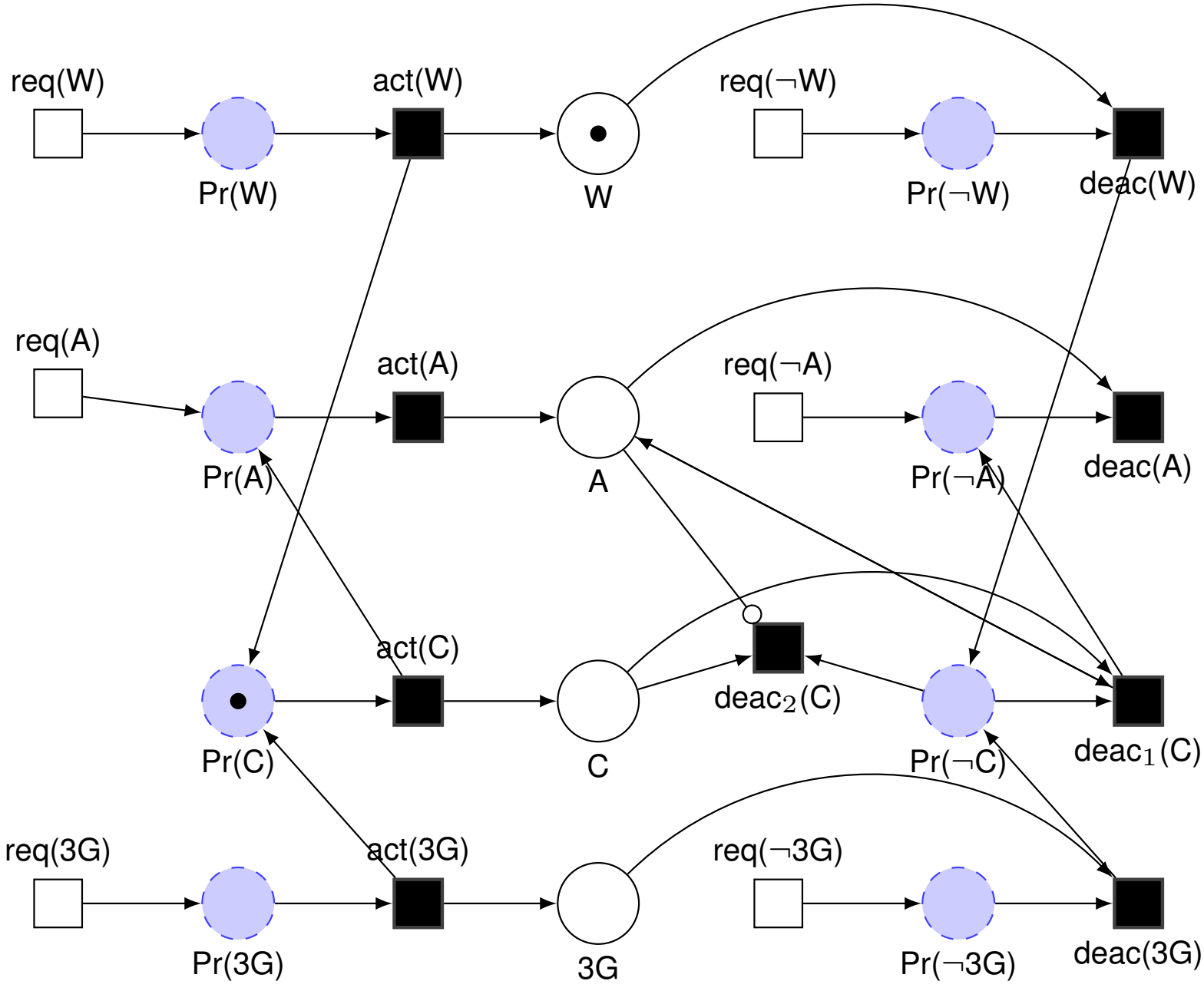
"The transition cannot be fired if there is token(s) in the place"

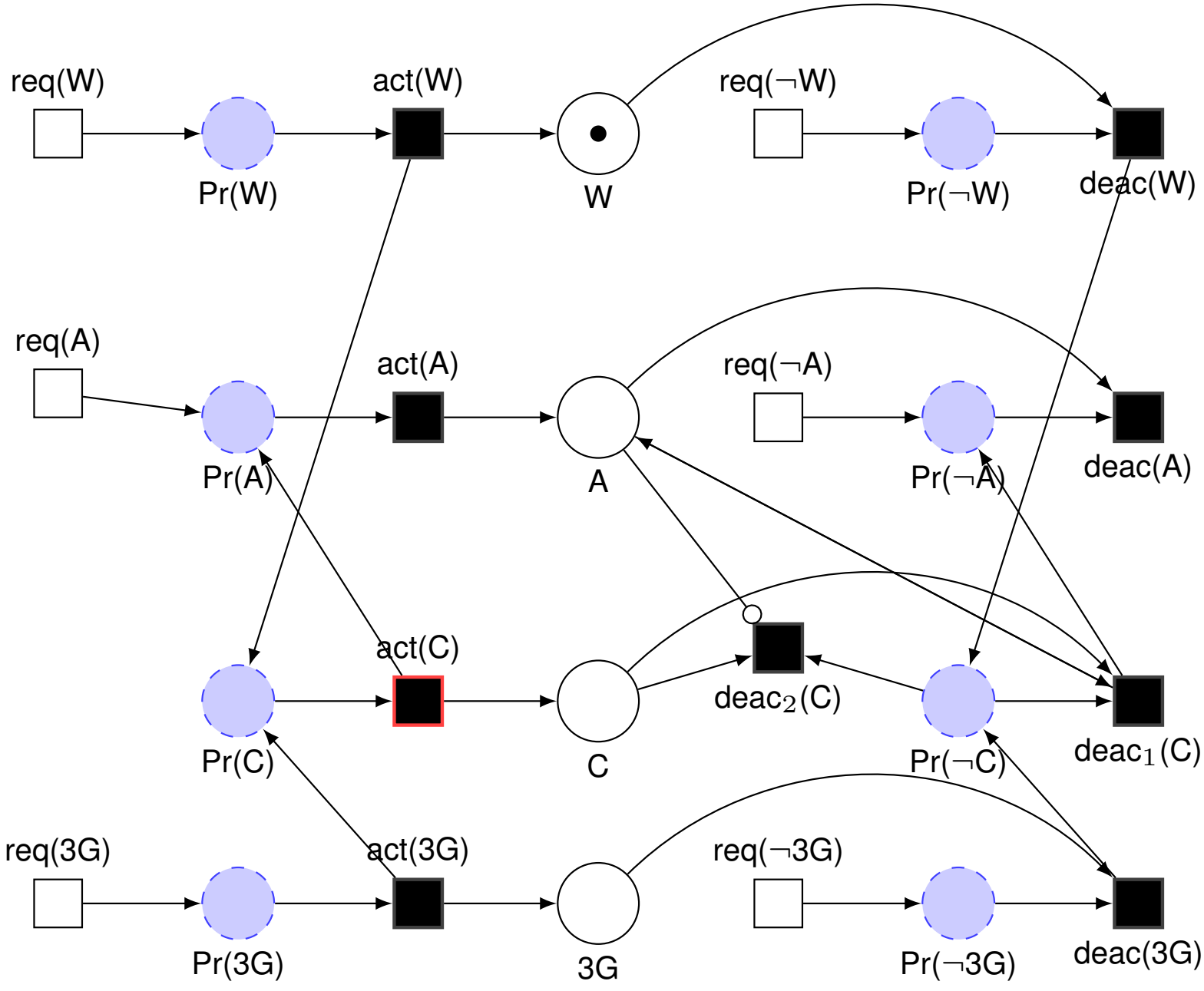

req(W)
act(W)
req(¬W)
Pr(W)
W
Pr(¬W)
deac(W)
req(A)
act(A)
req(¬A)
Pr(A)
A
Pr(¬A)
deac(A)
act(C)
deac2(C)
Pr(C)
C
Pr(¬C)
deac1(C)
req(3G)
act(3G)
req(¬3G)
Pr(3G)
3G
Pr(¬3G)
deac(3G)

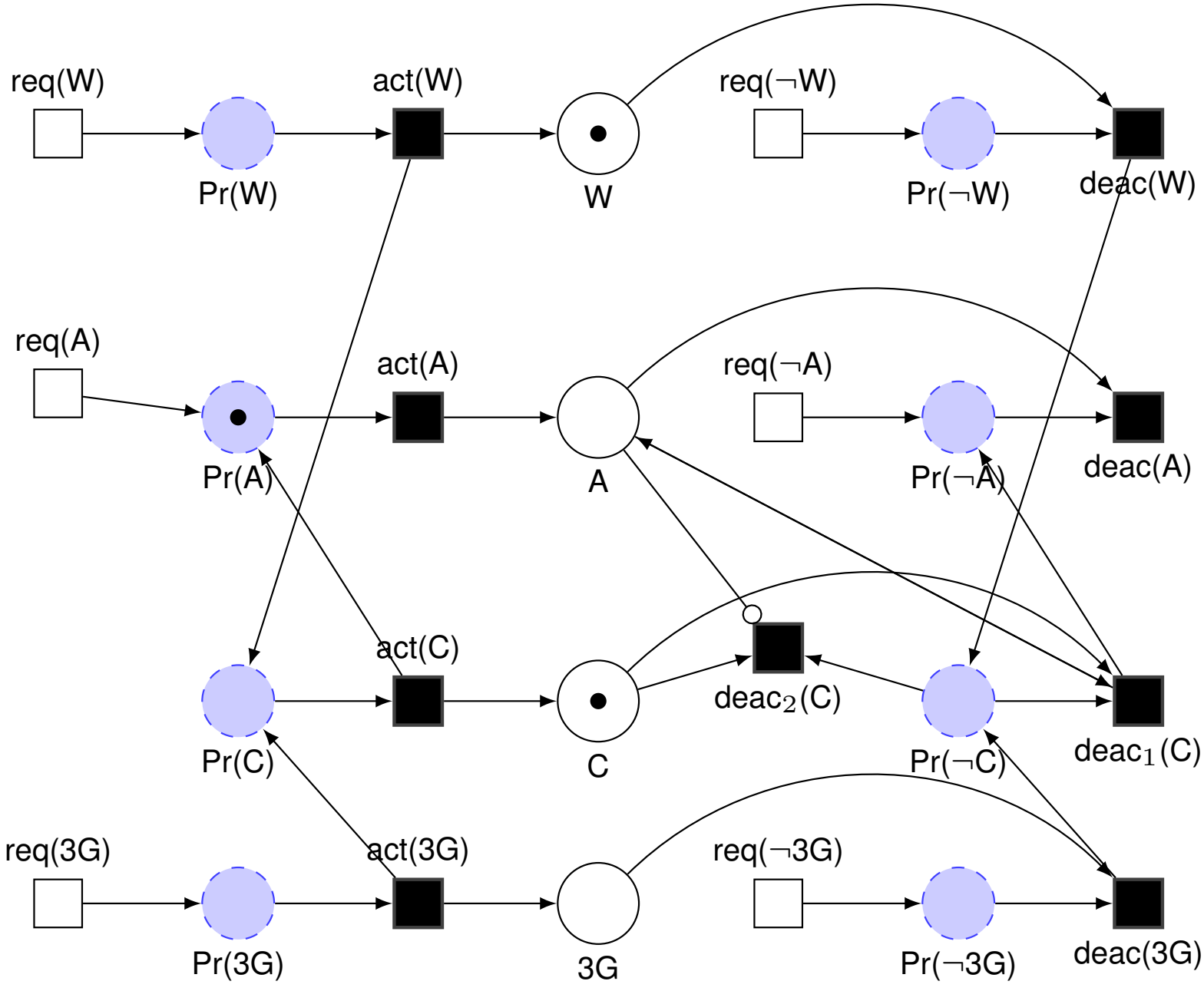

req(W)
Pr(W)
act(W)
W
req(¬W)
Pr(¬W)
deac(W)
req(A)
Pr(A)
act(A)
A
req(¬A)
Pr(¬A)
deac(A)
act(C)
C
deac2(C)
Pr(C)
Pr(¬C)
deac1(C)
req(3G)
Pr(3G)
act(3G)
3G
req(¬3G)
Pr(¬3G)
deac(3G)

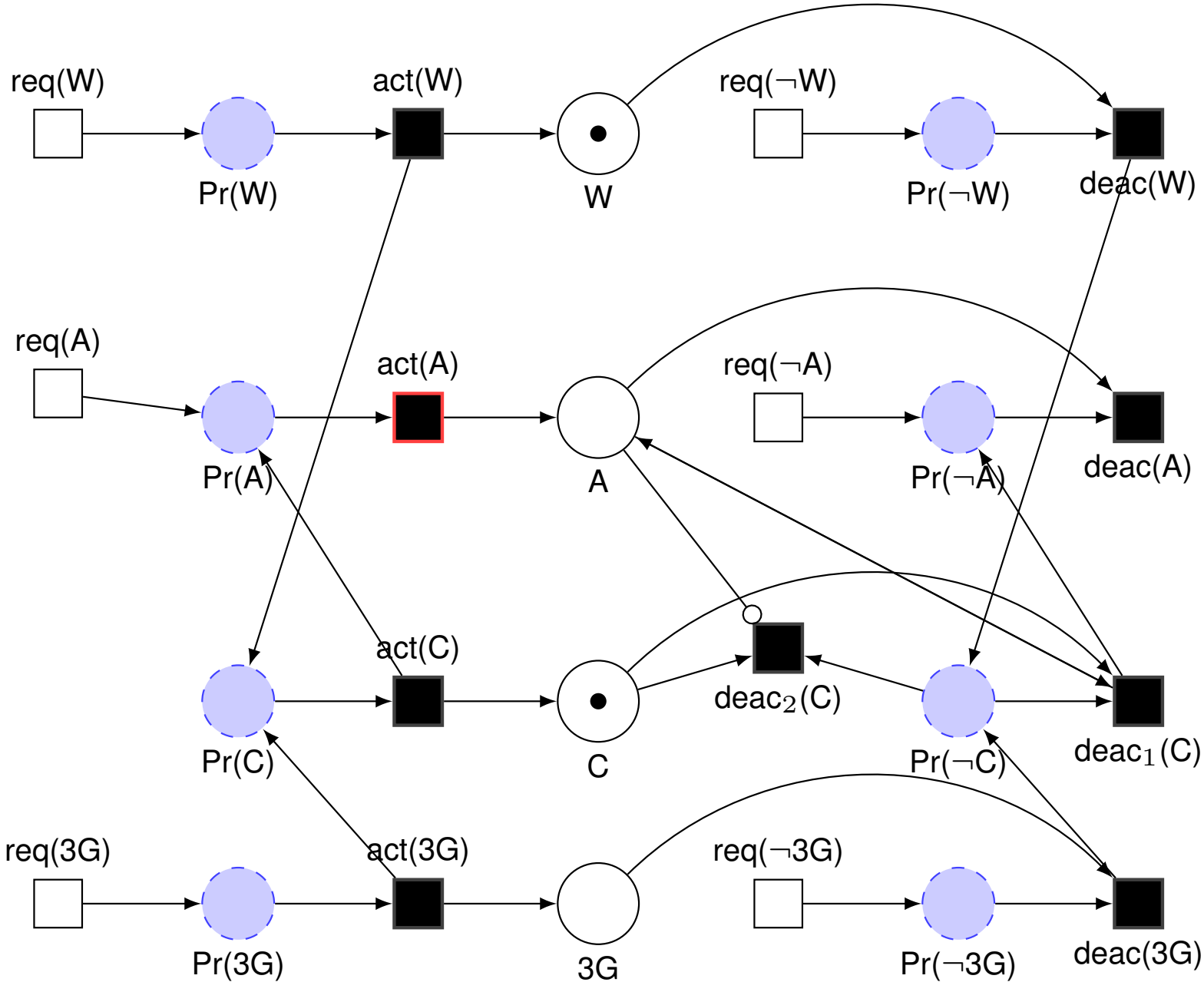

req(W)
act(W)
req(¬W)
Pr(W)
W
Pr(¬W)
deac(W)
req(A)
act(A)
req(¬A)
Pr(A)
A
Pr(¬A)
deac(A)
act(C)
deac2(C)
Pr(C)
C
Pr(¬C)
deac1(C)
req(3G)
act(3G)
req(¬3G)
Pr(3G)
3G
Pr(¬3G)
deac(3G)

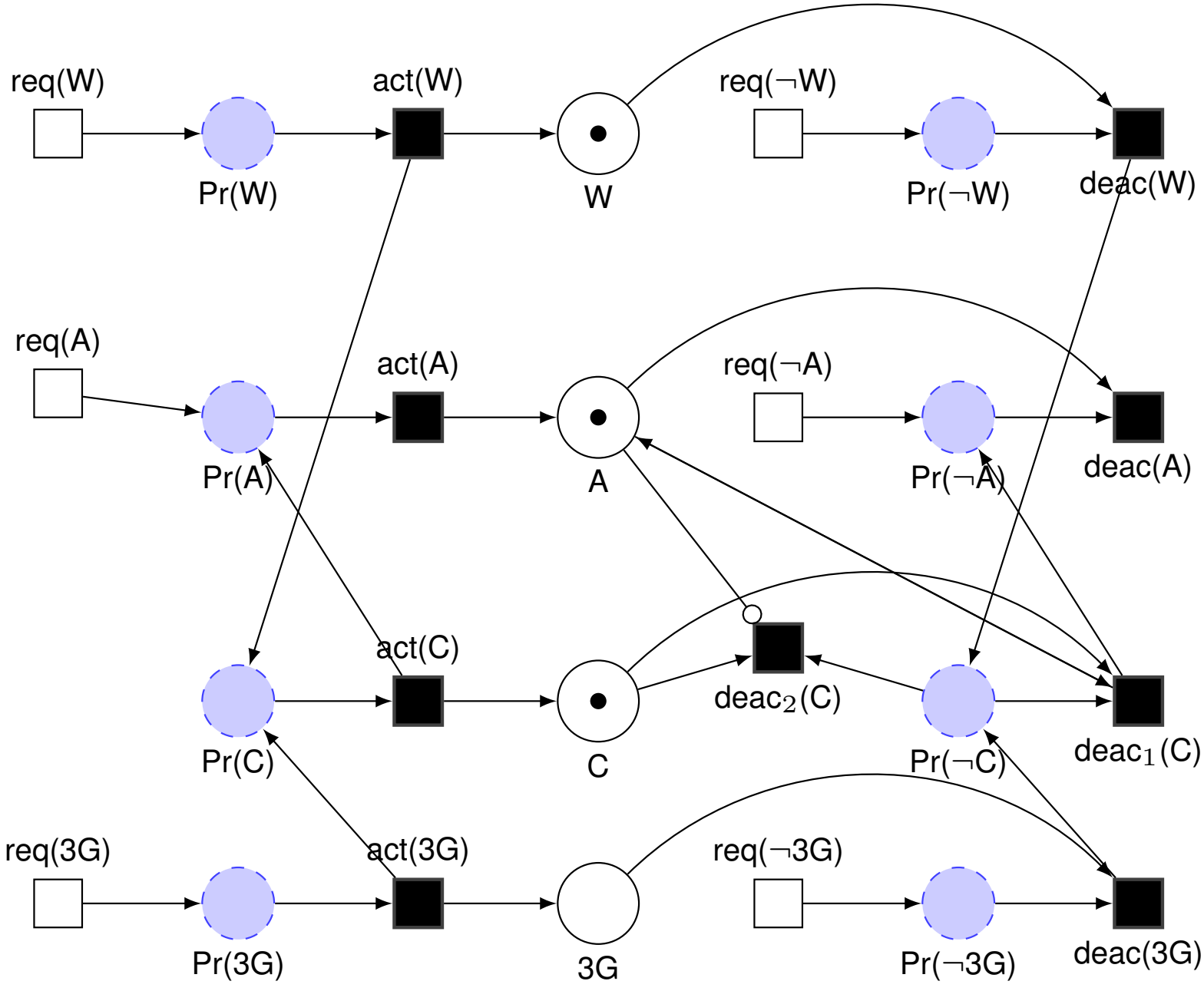

req(W)
act(W)
req(¬W)
Pr(W)
W
Pr(¬W)
deac(W)
req(A)
act(A)
req(¬A)
Pr(A)
A
Pr(¬A)
deac(A)
act(C)
deac2(C)
Pr(C)
C
Pr(¬C)
deac1(C)
req(3G)
act(3G)
req(¬3G)
Pr(3G)
3G
Pr(¬3G)
deac(3G)

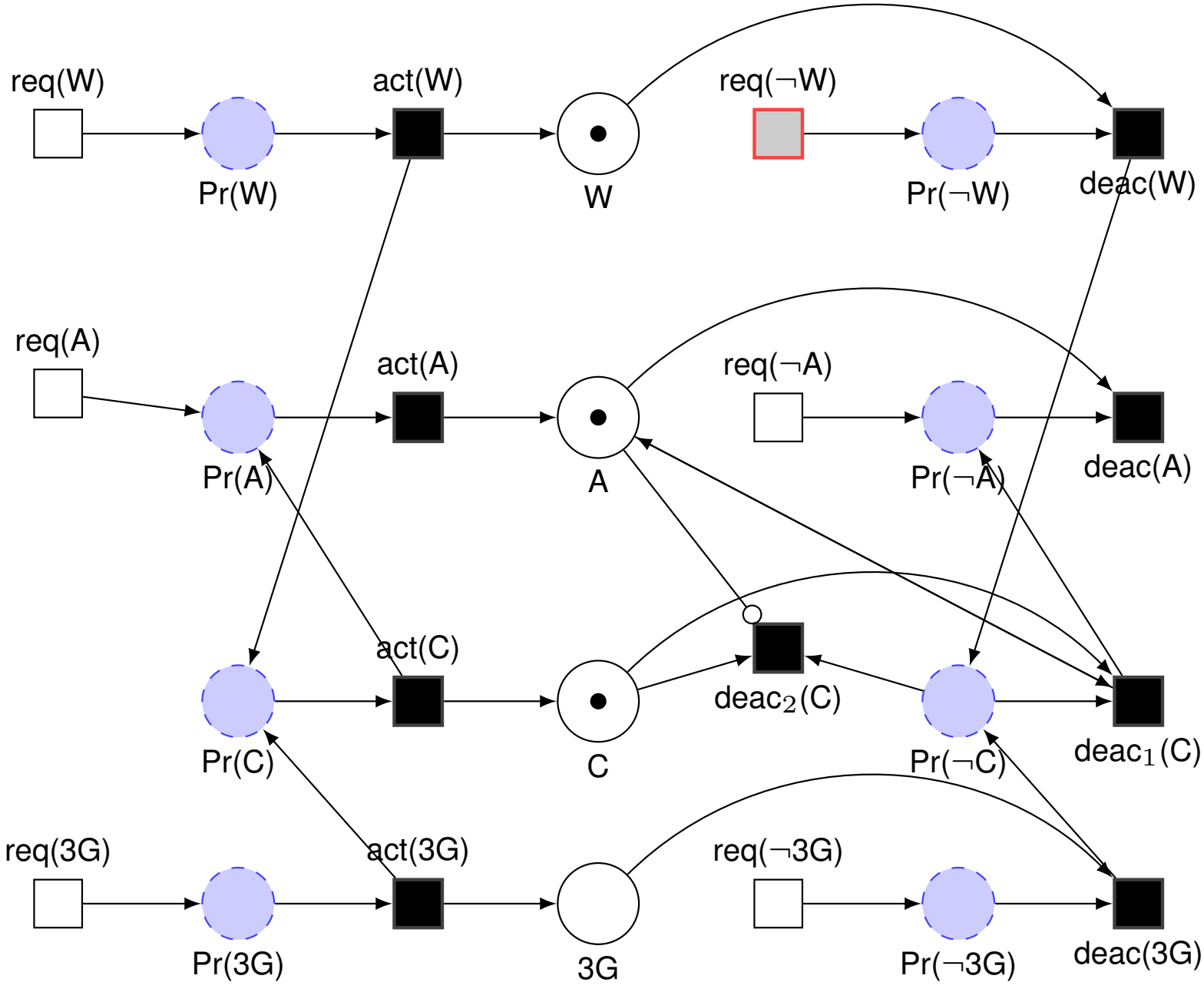

req(W)
act(W)
req(¬W)
Pr(W)
W
Pr(¬W)
deac(W)
req(A)
act(A)
req(¬A)
Pr(A)
A
Pr(¬A)
deac(A)
act(C)
deac2(C)
Pr(C)
C
Pr(¬C)
deac1(C)
req(3G)
act(3G)
req(¬3G)
Pr(3G)
3G
Pr(¬3G)
deac(3G)

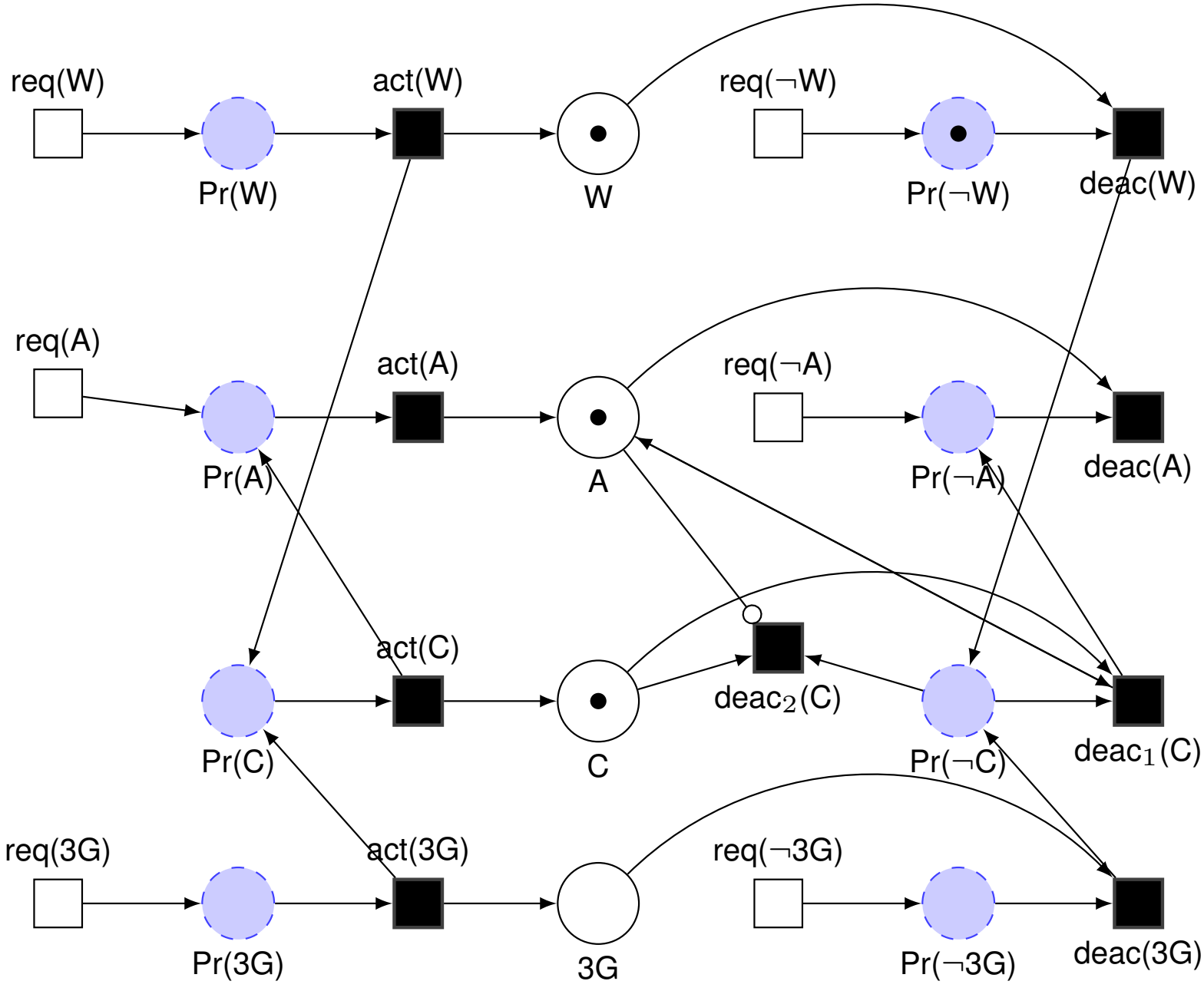

req(W)
act(W)
req(¬W)
Pr(W)
W
Pr(¬W)
deac(W)
req(A)
act(A)
req(¬A)
Pr(A)
A
Pr(¬A)
deac(A)
act(C)
deac2(C)
Pr(C)
C
Pr(¬C)
deac1(C)
req(3G)
act(3G)
req(¬3G)
Pr(3G)
3G
Pr(¬3G)
deac(3G)

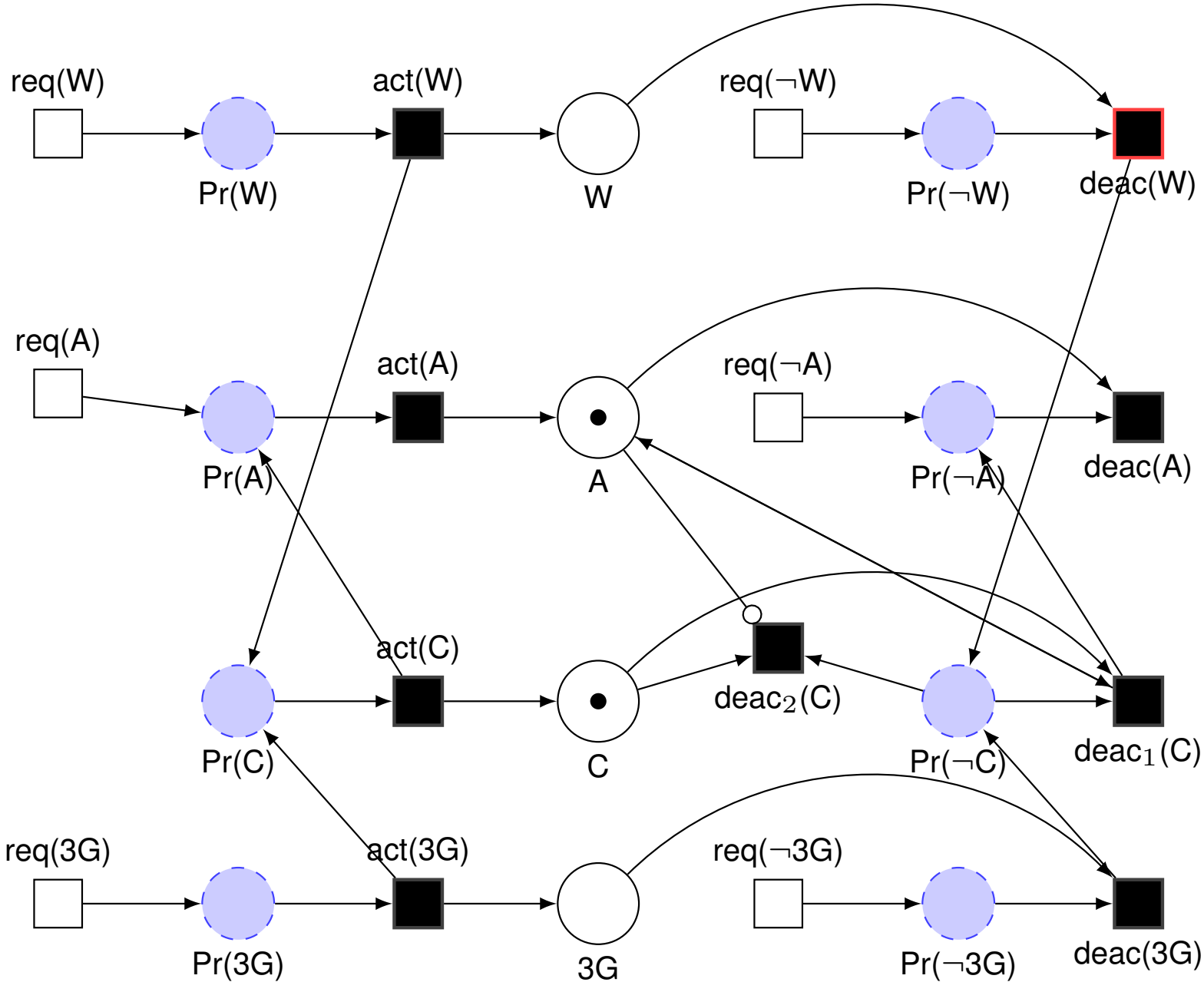

req(W)
act(W)
req(¬W)
Pr(W)
W
Pr(¬W)
deac(W)
req(A)
act(A)
req(¬A)
Pr(A)
A
Pr(¬A)
deac(A)
act(C)
deac2(C)
Pr(C)
C
Pr(¬C)
deac1(C)
req(3G)
act(3G)
req(¬3G)
Pr(3G)
3G
Pr(¬3G)
deac(3G)

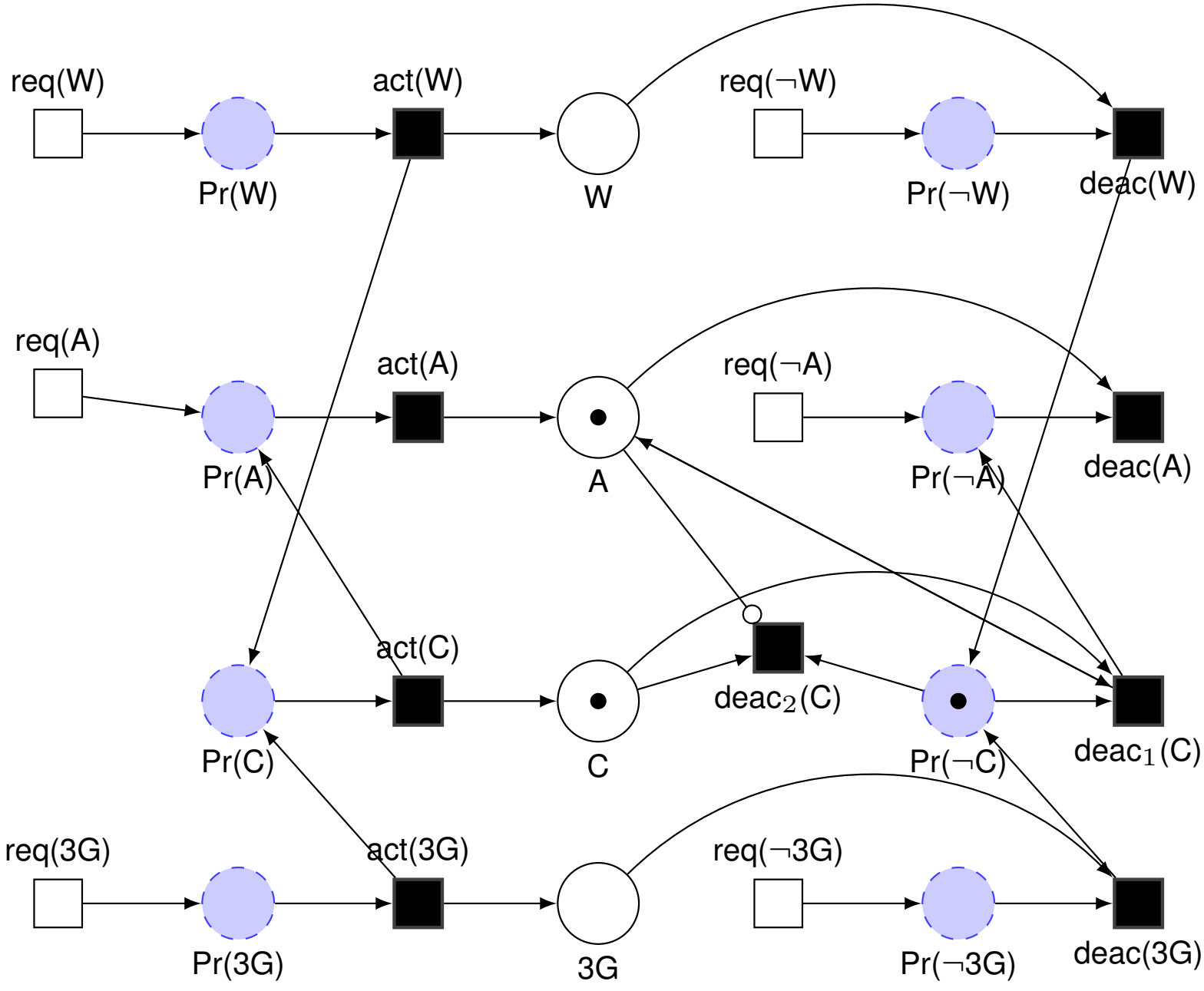

req(W)
act(W)
req(¬W)
Pr(W)
W
Pr(¬W)
deac(W)
req(A)
act(A)
req(¬A)
Pr(A)
A
Pr(¬A)
deac(A)
act(C)
deac2(C)
Pr(C)
C
Pr(¬C)
deac1(C)
req(3G)
act(3G)
req(¬3G)
Pr(3G)
3G
Pr(¬3G)
deac(3G)

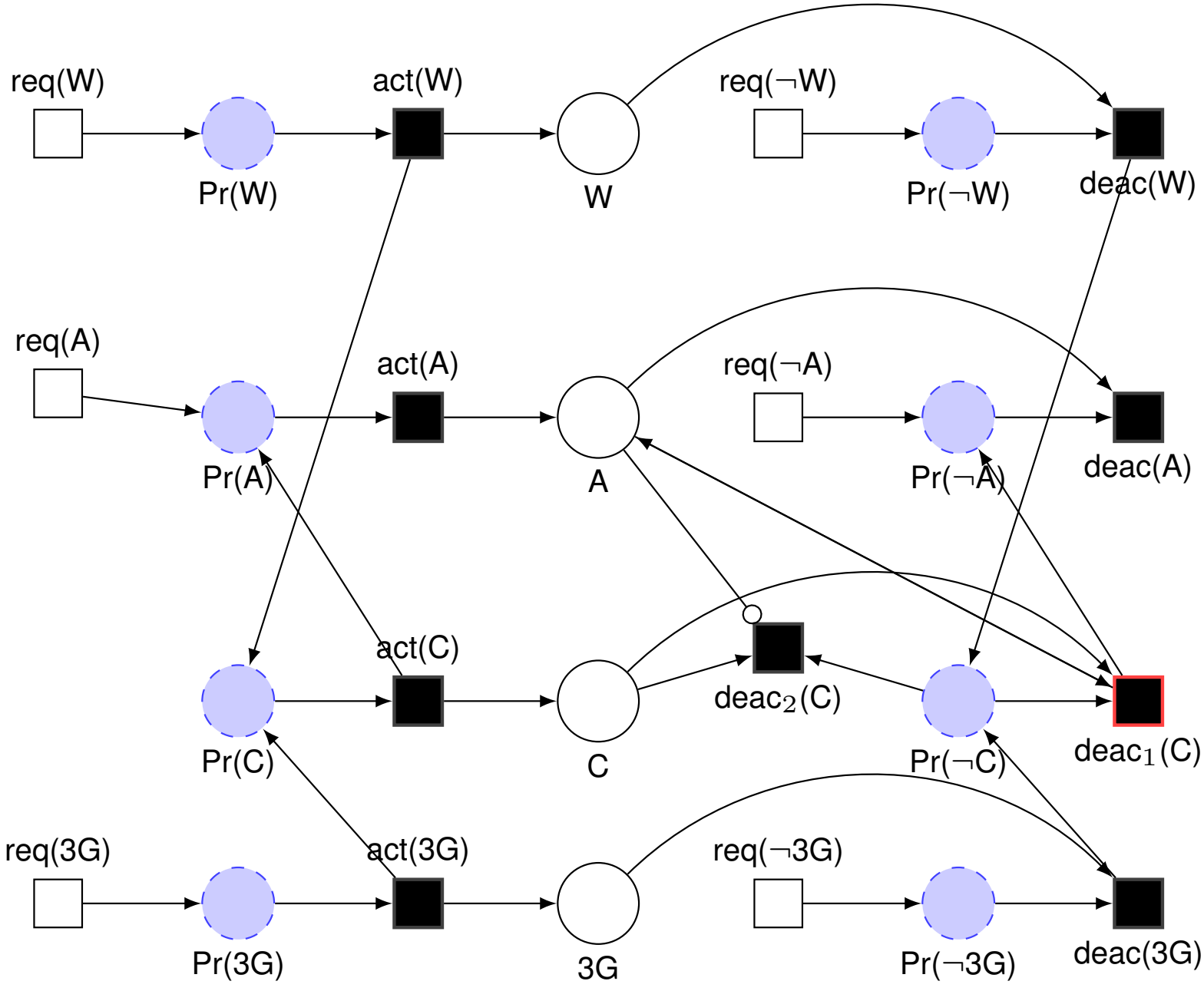

req(W)
act(W)
req(¬W)
Pr(W)
W
Pr(¬W)
deac(W)
req(A)
act(A)
req(¬A)
Pr(A)
A
Pr(¬A)
deac(A)
act(C)
deac$_2$(C)
Pr(C)
C
Pr(¬C)
deac$_1$(C)
req(3G)
act(3G)
req(¬3G)
Pr(3G)
3G
Pr(¬3G)
deac(3G)

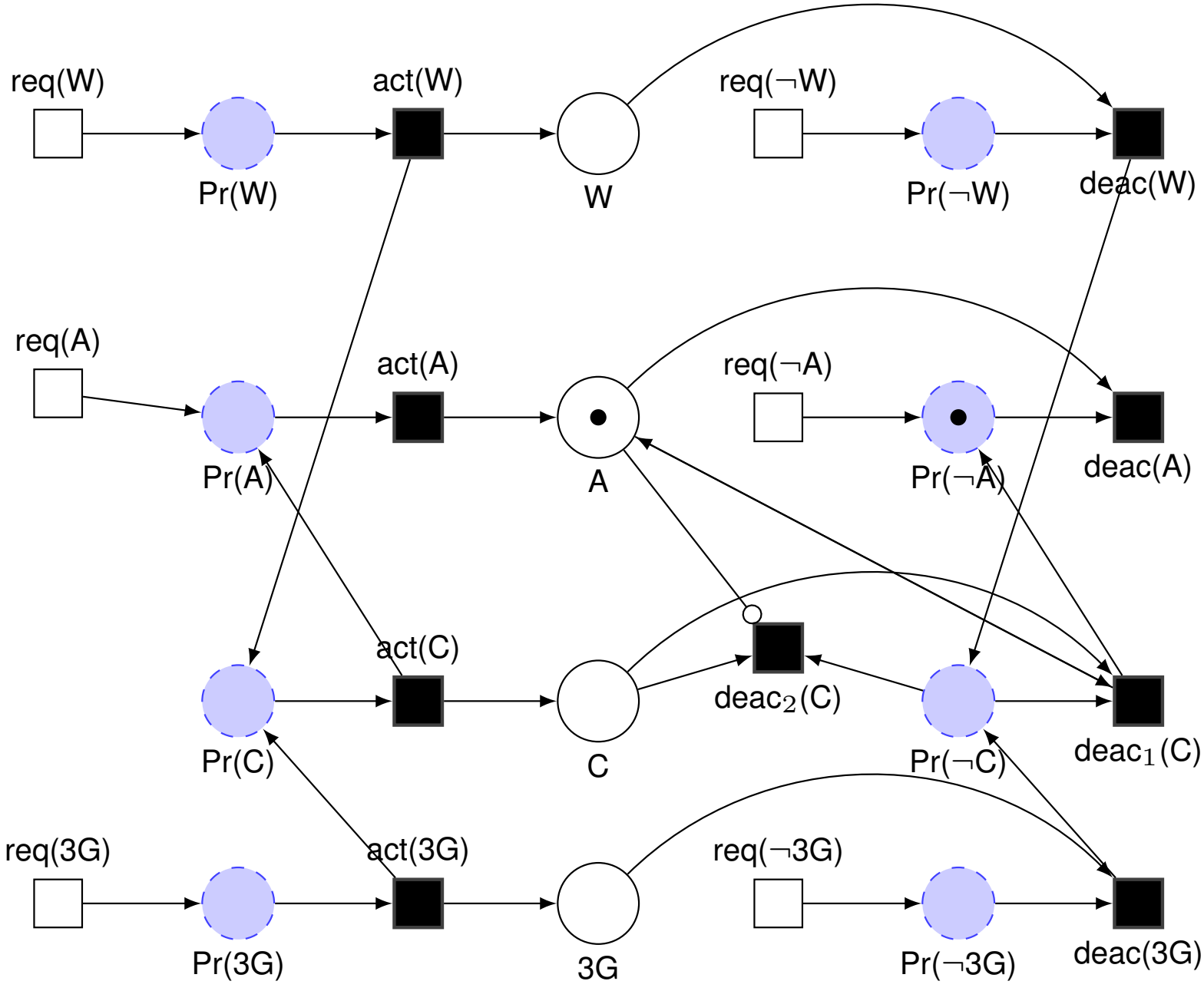

req(W)
act(W)
req(¬W)
Pr(W)
W
Pr(¬W)
deac(W)
req(A)
act(A)
req(¬A)
Pr(A)
A
Pr(¬A)
deac(A)
act(C)
deac2(C)
Pr(C)
C
Pr(¬C)
deac1(C)
req(3G)
act(3G)
req(¬3G)
Pr(3G)
3G
Pr(¬3G)
deac(3G)

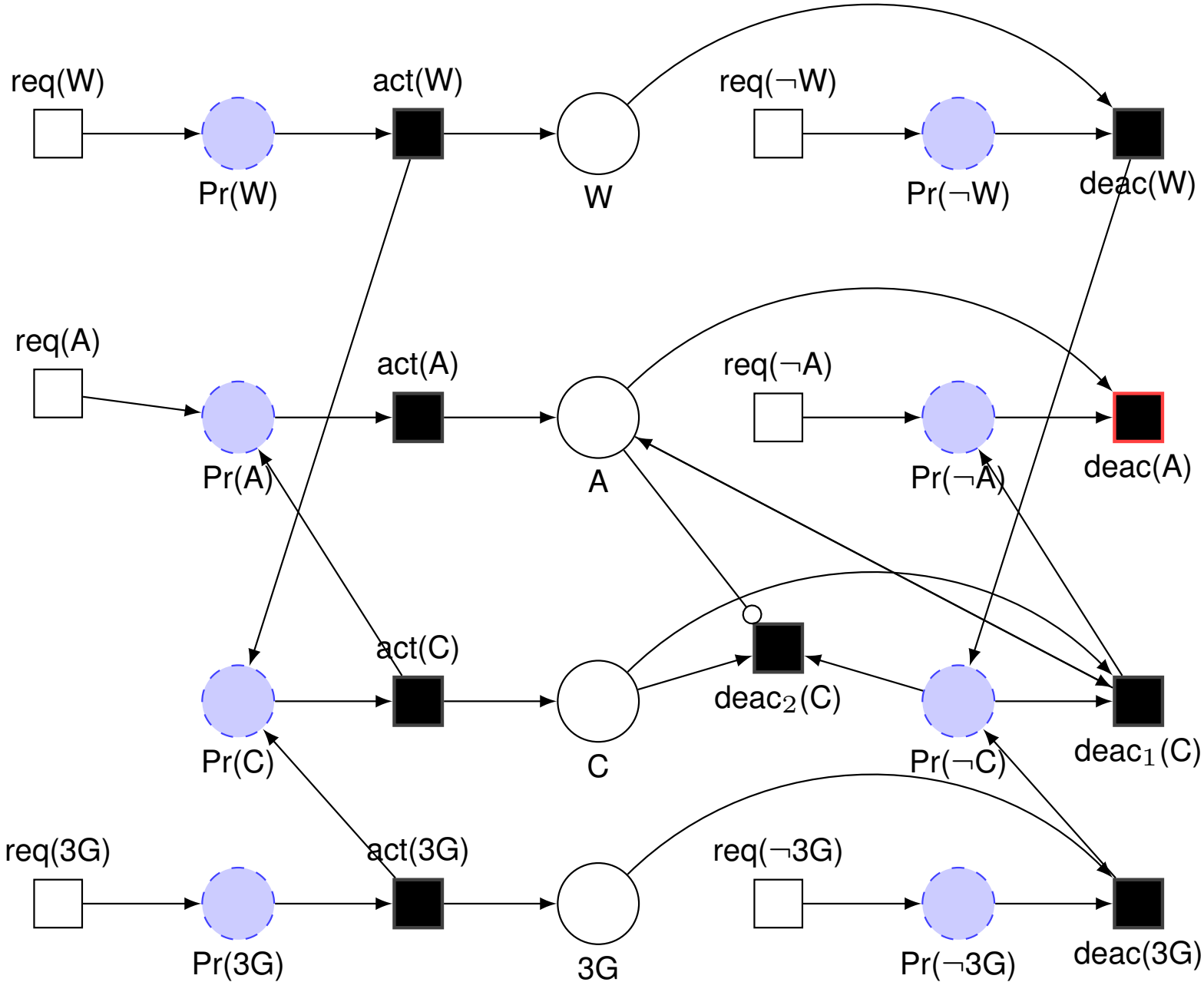

req(W)
act(W)
req(¬W)
Pr(W)
W
Pr(¬W)
deac(W)
req(A)
act(A)
req(¬A)
Pr(A)
A
Pr(¬A)
deac(A)
act(C)
deac2(C)
Pr(C)
C
Pr(¬C)
deac1(C)
req(3G)
act(3G)
req(¬3G)
Pr(3G)
3G
Pr(¬3G)
deac(3G)

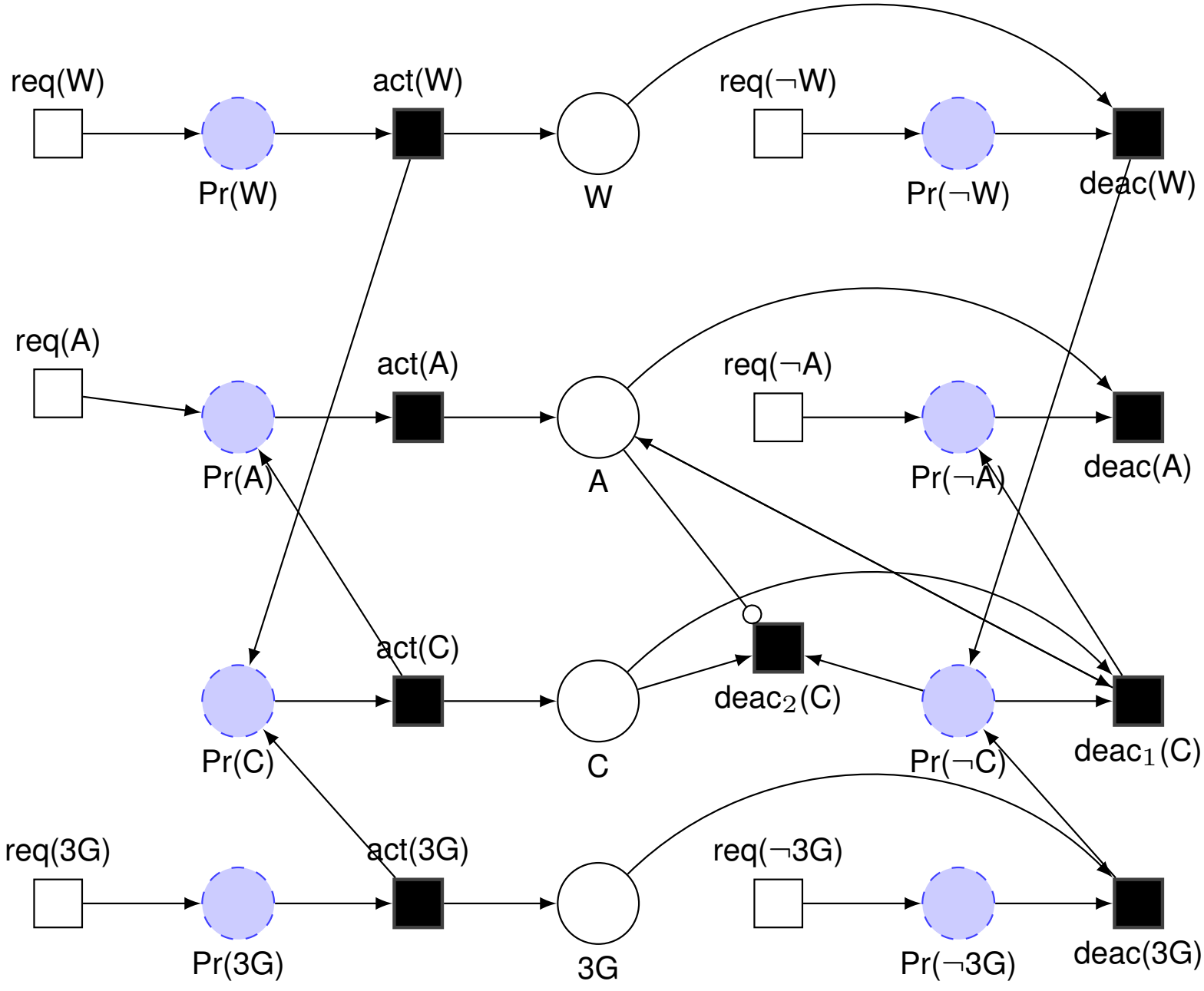

req(W)
act(W)
req(¬W)
Pr(W)
W
Pr(¬W)
deac(W)
req(A)
act(A)
req(¬A)
Pr(A)
A
Pr(¬A)
deac(A)
act(C)
deac2(C)
Pr(C)
C
Pr(¬C)
deac1(C)
req(3G)
act(3G)
req(¬3G)
Pr(3G)
3G
Pr(¬3G)
deac(3G)


req(W)
act(W)
req(¬W)
Pr(W)
W
Pr(¬W)
deac(W)
req(A)
act(A)
req(¬A)
Pr(A)
A
Pr(¬A)
deac(A)
act(C)
deac2(C)
Pr(C)
C
Pr(¬C)
deac1(C)
req(3G)
act(3G)
req(¬3G)
Pr(3G)
3G
Pr(¬3G)
deac(3G)


req(W)
Pr(W)
act(W)
W
req(¬W)
Pr(¬W)
deac(W)
req(A)
Pr(A)
act(A)
A
req(¬A)
Pr(¬A)
deac(A)
act(C)
Pr(C)
C
deac2(C)
Pr(¬C)
deac1(C)
req(3G)
Pr(3G)
act(3G)
3G
req(¬3G)
Pr(¬3G)
deac(3G)


req(W)
Pr(W)
act(W)
W
req(¬W)
Pr(¬W)
deac(W)
req(A)
Pr(A)
act(A)
A
req(¬A)
Pr(¬A)
deac(A)
act(C)
Pr(C)
C
deac2(C)
Pr(¬C)
deac1(C)
req(3G)
Pr(3G)
act(3G)
3G
req(¬3G)
Pr(¬3G)
deac(3G)


req(W)
Pr(W)
act(W)
W
req(¬W)
Pr(¬W)
deac(W)
req(A)
Pr(A)
act(A)
A
req(¬A)
Pr(¬A)
deac(A)
act(C)
C
deac2(C)
Pr(C)
Pr(¬C)
deac1(C)
req(3G)
Pr(3G)
act(3G)
3G
req(¬3G)
Pr(¬3G)
deac(3G)


req(W)
act(W)
req(¬W)
Pr(W)
W
Pr(¬W)
deac(W)
req(A)
act(A)
req(¬A)
Pr(A)
A
Pr(¬A)
deac(A)
act(C)
deac2(C)
Pr(C)
C
Pr(¬C)
deac1(C)
req(3G)
act(3G)
req(¬3G)
Pr(3G)
3G
Pr(¬3G)
deac(3G)

Context Petri Nets (CoPN)

The previous Petri Net is called a **Context Petri Net**.

There are many more relations between contexts:

- Conjunction dependency relation
- Suggestion dependency relation
- Requirement dependency relation
- Implication dependency relation
- ...

There is a corresponding semantic rule for each one in Context Petri Nets!

Context Petri Nets (CoPN) in practice

Subjective-C has built-in language constructs to interact with CoPNs.

The CoPN is automatically generated based on the declaration of contexts and context dependency relations.

Coverability, reachability and liveness properties of Petri Nets can also be verified on CoPNs.

Usefulness of Context Petri Nets

CoPN provides a formalism for semantics of context activation and deactivation to make sure that subtle errors are unnoticed in context-oriented systems.

In Subjective-C

Researchers found an error in the Subjective-C implementation of the *implies* relation.

Two different contexts can imply the activation of the same context. In the original definition of the relation in Subjective-C, the deactivation of one of those context led to the **full** deactivation of the implied context. This is the kind of subtle errors that you cannot discover if you don't have a strong practical formalism that supports relations and multiple activations of contexts.

Plan

1. Introduction

2. Context-oriented programming

3. Context modelling

4. Conclusion

Conclusion

- “Standard” Petri Nets are great at modelling context (de)activations
- But they are not expressive enough for dependency relations
- Problems can be solved by adding extensions to Petri Nets
- Context Petri Nets help to verify semantics of context activation in context-oriented systems
- Tools can perform automated model checking given the dependency relations

Conclusion

- “Standard” Petri Nets are great at modelling context (de)activations
- But they are not expressive enough for dependency relations
- Problems can be solved by adding extensions to Petri Nets
- Context Petri Nets help to verify semantics of context activation in context-oriented systems
- Tools can perform automated model checking given the dependency relations

Conclusion

- “Standard” Petri Nets are great at modelling context (de)activations
- But they are not expressive enough for dependency relations
- Problems can be solved by adding extensions to Petri Nets
- Context Petri Nets help to verify semantics of context activation in context-oriented systems
- Tools can perform automated model checking given the dependency relations

Conclusion

- “Standard” Petri Nets are great at modelling context (de)activations
- But they are not expressive enough for dependency relations
- Problems can be solved by adding extensions to Petri Nets
- Context Petri Nets help to verify semantics of context activation in context-oriented systems
- Tools can perform automated model checking given the dependency relations

Conclusion

- “Standard” Petri Nets are great at modelling context (de)activations
- But they are not expressive enough for dependency relations
- Problems can be solved by adding extensions to Petri Nets
- Context Petri Nets help to verify semantics of context activation in context-oriented systems
- Tools can perform automated model checking given the dependency relations

Questions?

Exclusion between contexts

Example A system with 2 contexts:

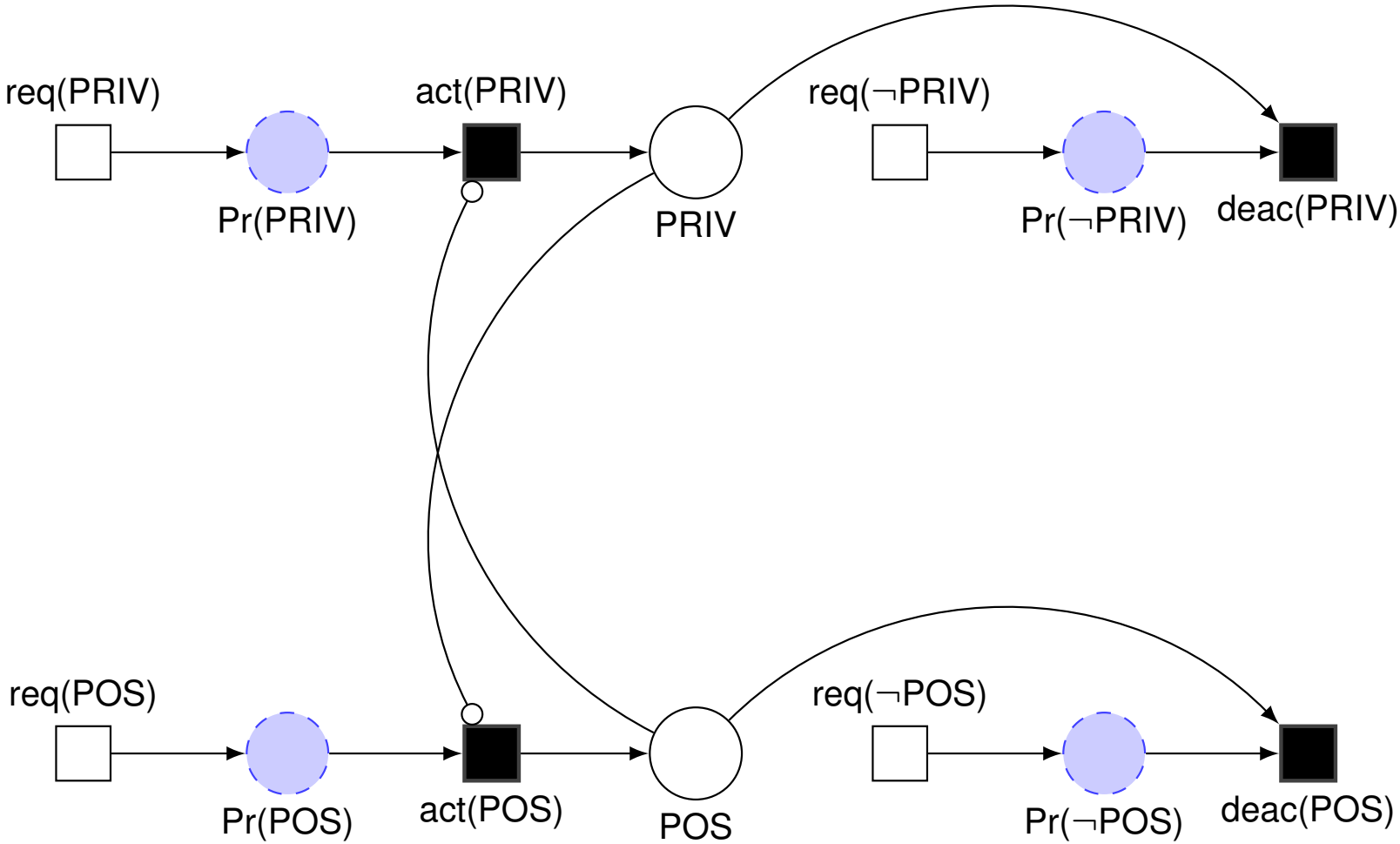
- Private context (Priv)
- Positioning context (Pos)

If the user doesn't want to share private information, you can't activate the positioning.

Relation type

This conflicting behavior is called an *exclusion dependency relation*, denoted Priv □-□ Pos

Exclusion dependency relation



Implication dependency relation

Example A system with 2 contexts:

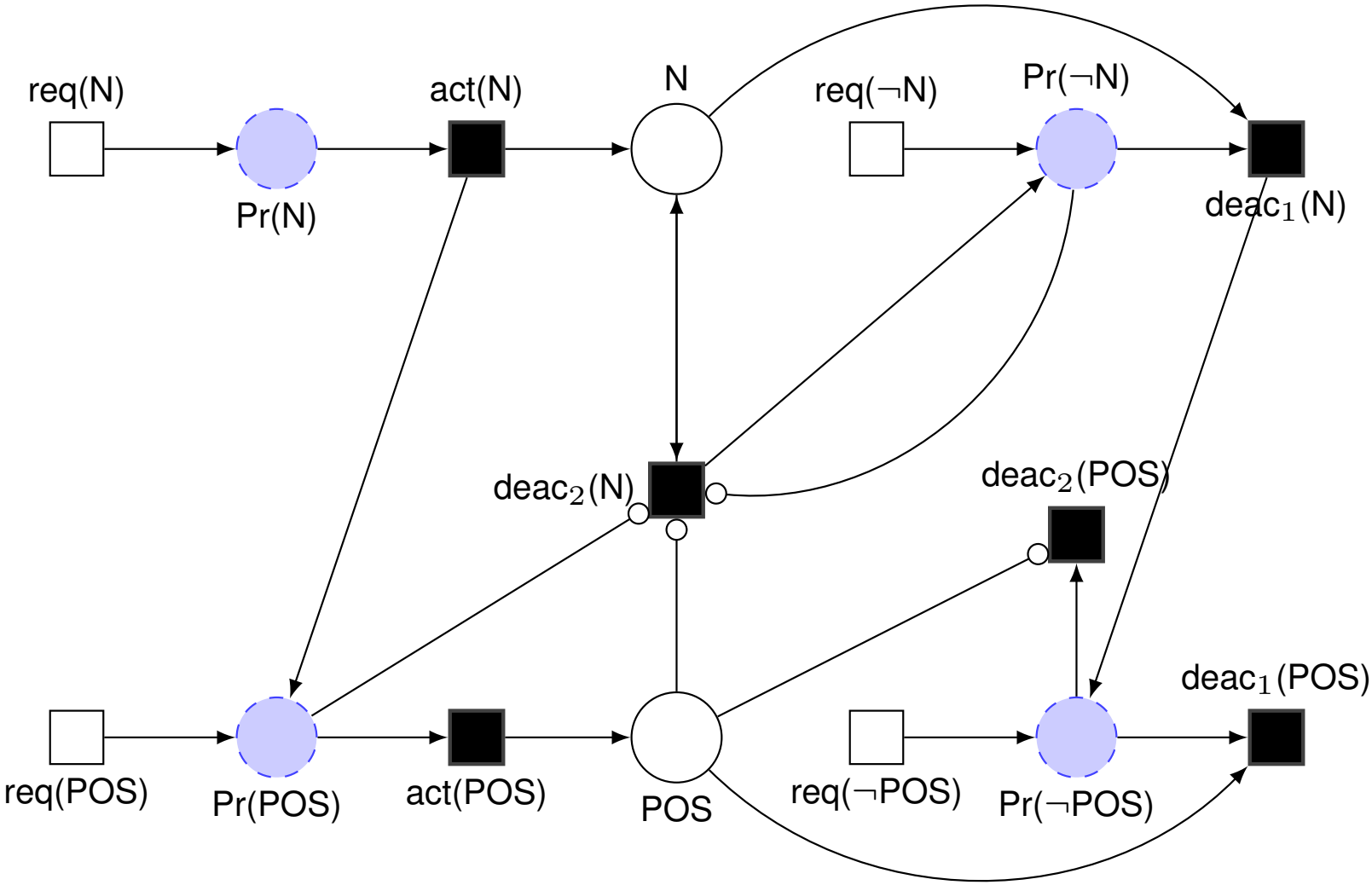
- NLBS context (N)
- Positioning context (Pos)

NLBS provides services that are explicitly used by Positioning context. Conversely, if the Positioning is deactivated, the services of the NLBS are no longer needed.

Relation type

This containment interaction is called an *implication dependency relation*, denoted $\text{N} \longrightarrow\!\blacktriangleright \text{Pos}$

Implication dependency relation



Requirement dependency relation

Example A system with two contexts:

- NLBS context (N)
- Connectivity context (C)

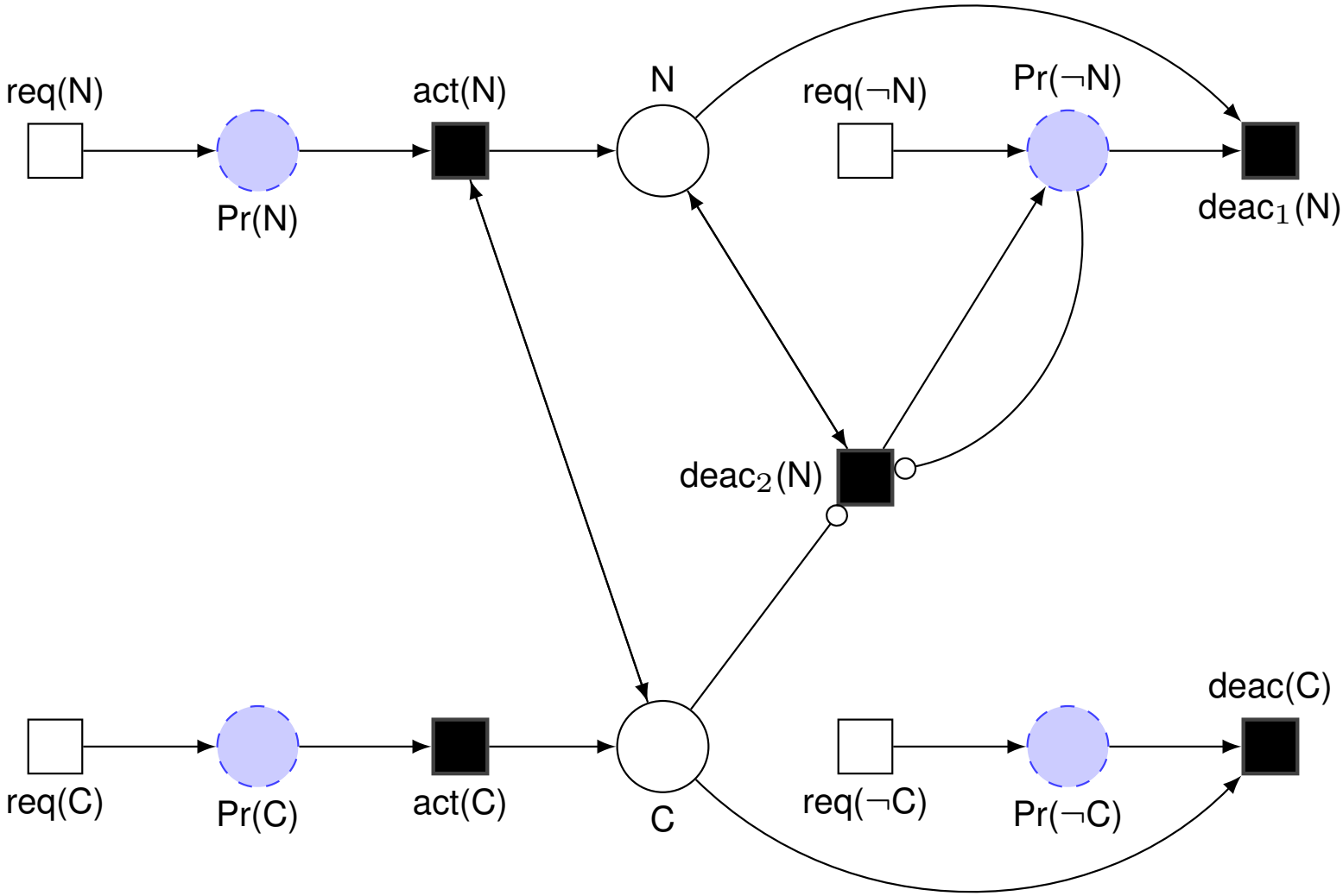
The position calculation of the NLBS service is based on the location inferred from a local network connection. The source context NLBS *requires* the target context Connectivity.

Relation type

This relation between an activation of a context and an already active context is called a *requirement dependency relation*, denoted

N ⟵◀ Pos

Requirement dependency relation



Suggestion dependency relation

Example A system (on a phone) with 3 contexts:

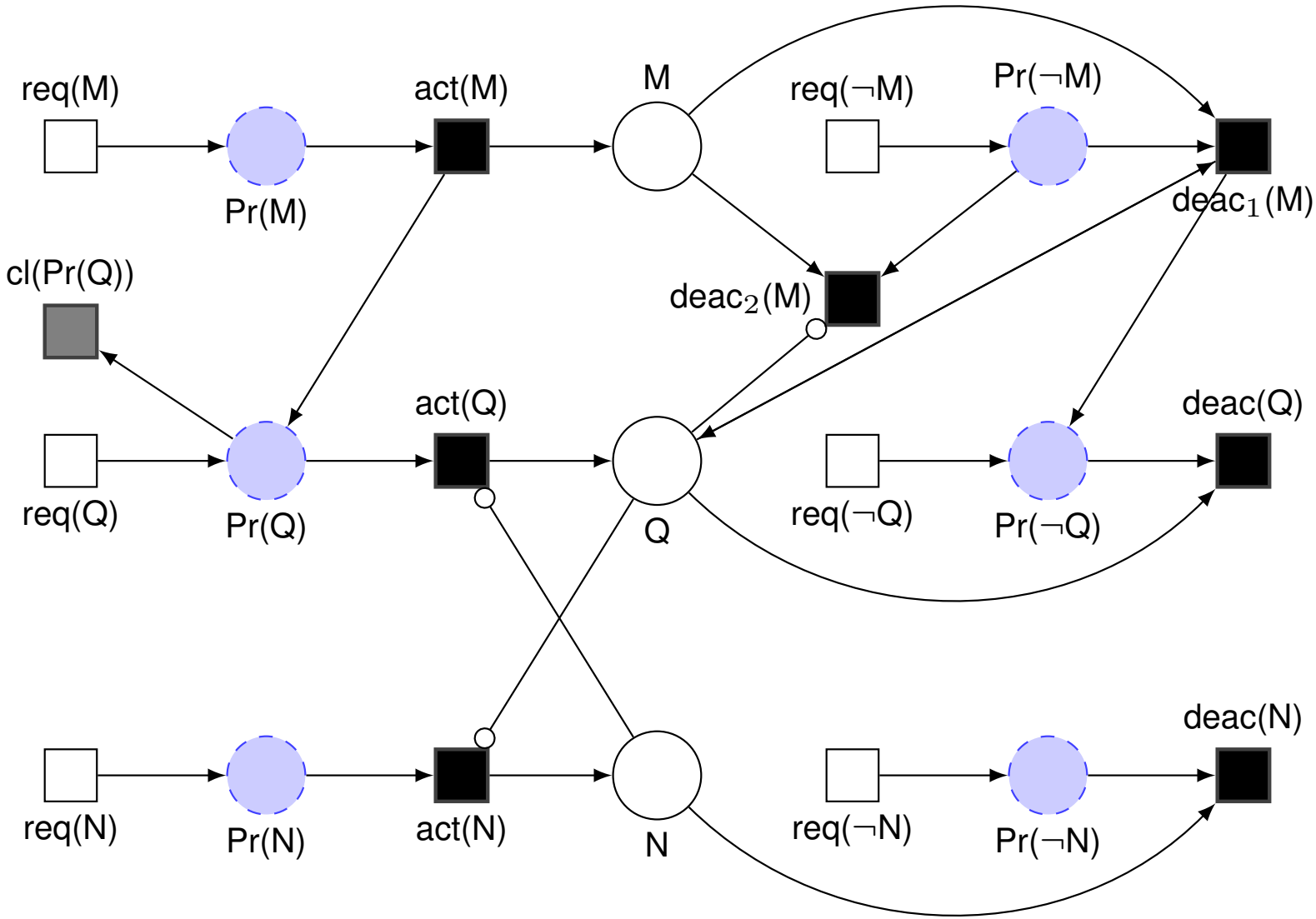
- Meeting context (M)
- Quiet context (Q)
- Noisy context (N)

When the user enters a meeting room, the Meeting context is activated. Meetings should be quiet, so incoming calls should be quiet. But this is not a strong constraint: meetings can become quite noisy.

Relation type

There is an exclusion dependency relation between Quiet and Noisy, and there is a *suggestion dependency relation* between Meeting and Quiet, denoted $M \dashrightarrow Q$

Suggestion dependency relation



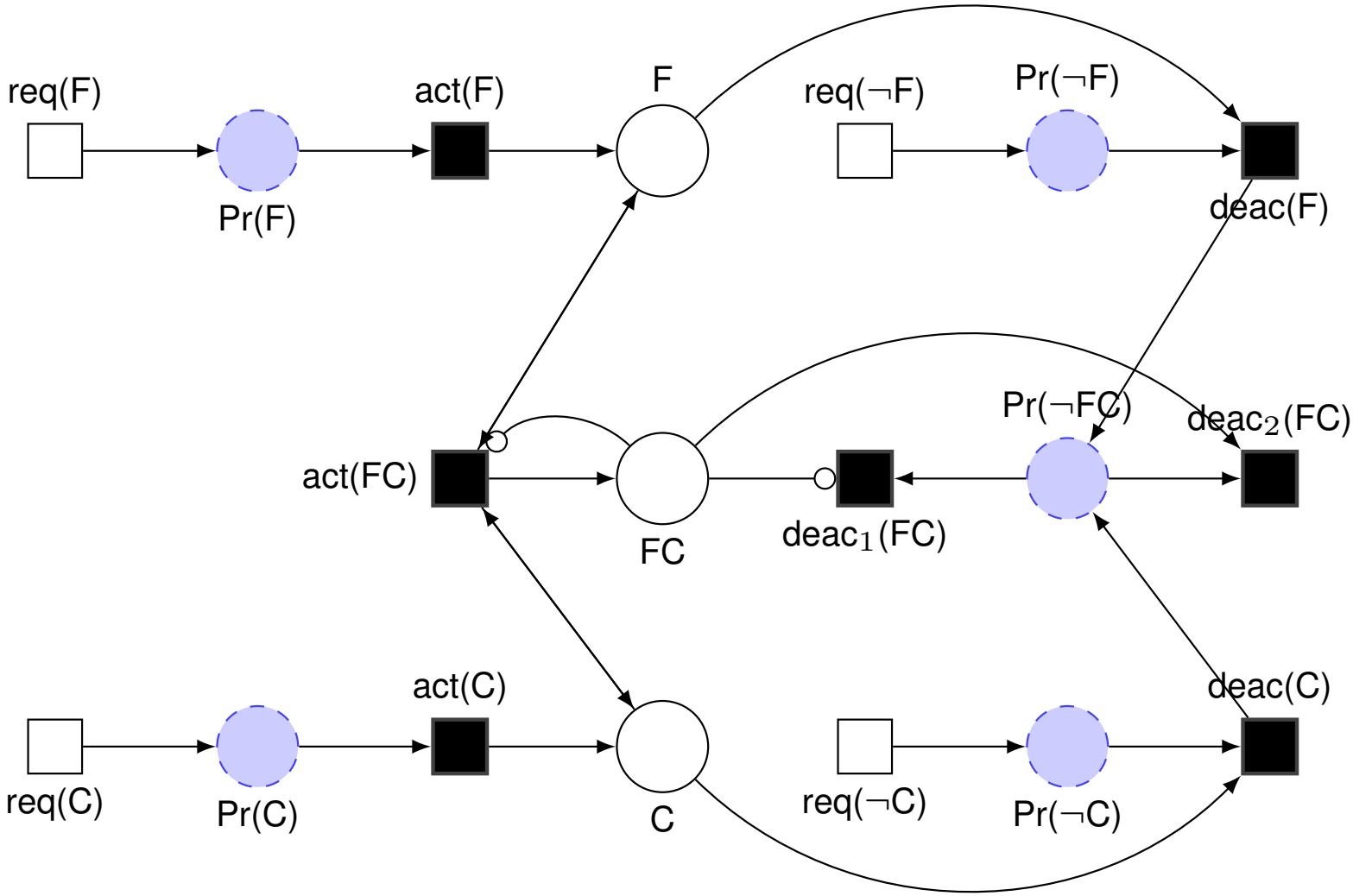
Conjunction dependency relation

Example A system with 2 contexts:

- Friends context (F)
- Connectivity context (C)

When the both contexts are activated, additional functionality is made available.

Relation type

This relation between the contexts is called a *conjunction dependency relation*, denoted $\rightarrow$(F C).

Conjunction dependency relation