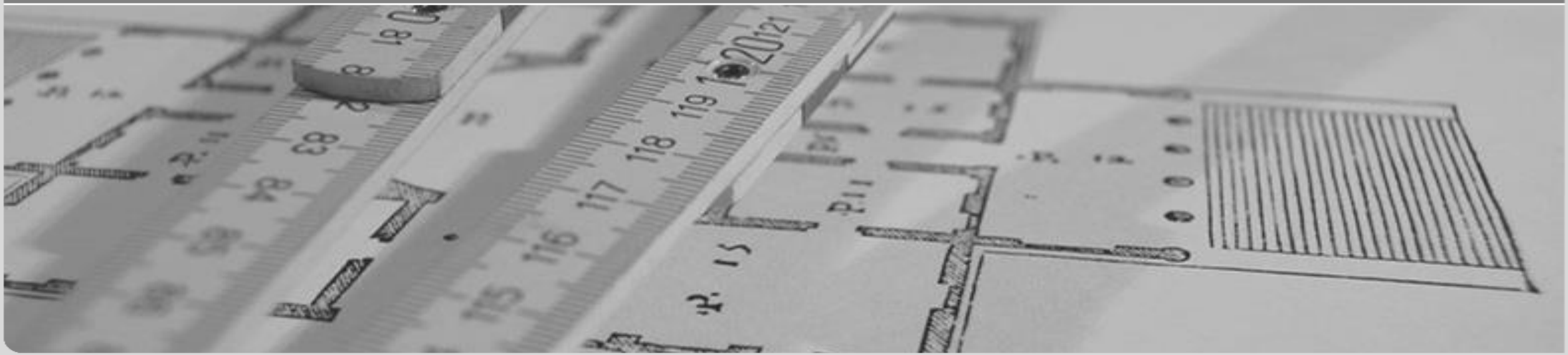
Leveraging State to Facilitate Separation of Concerns in Reuse-oriented Performance Models

Dominik Werle, Stephan Seifermann, Sebastian D. Krach

8th Symposium on Software Performance 2017

ARCHITECTURE-DRIVEN REQUIREMEMTS ENGINEERING GROUP
INSTITUTE FOR PROGRAM STRUCTURES AND DATA ORGANIZATION, FACULTY OF INFORMATICS



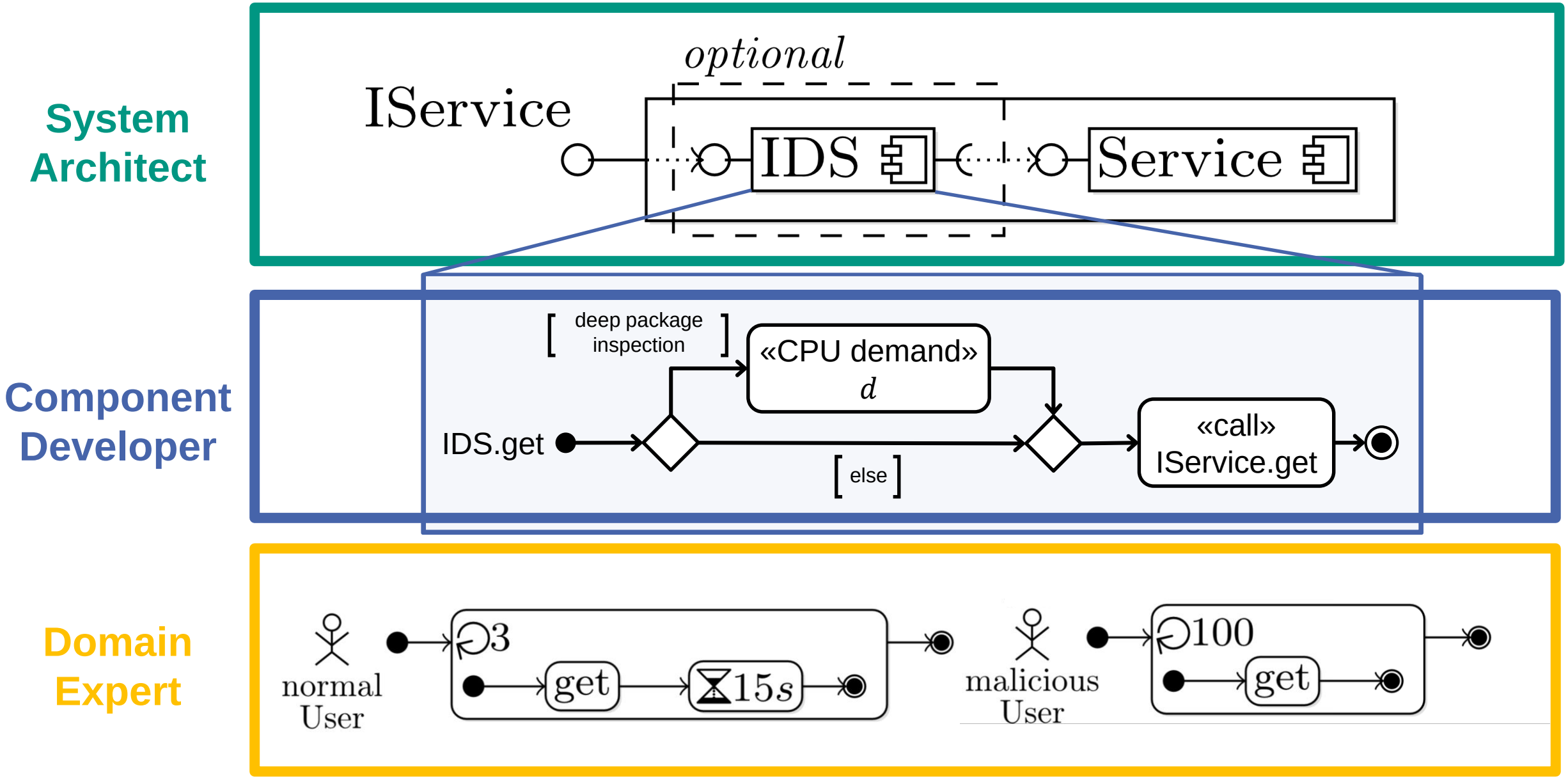
Contribution Overview

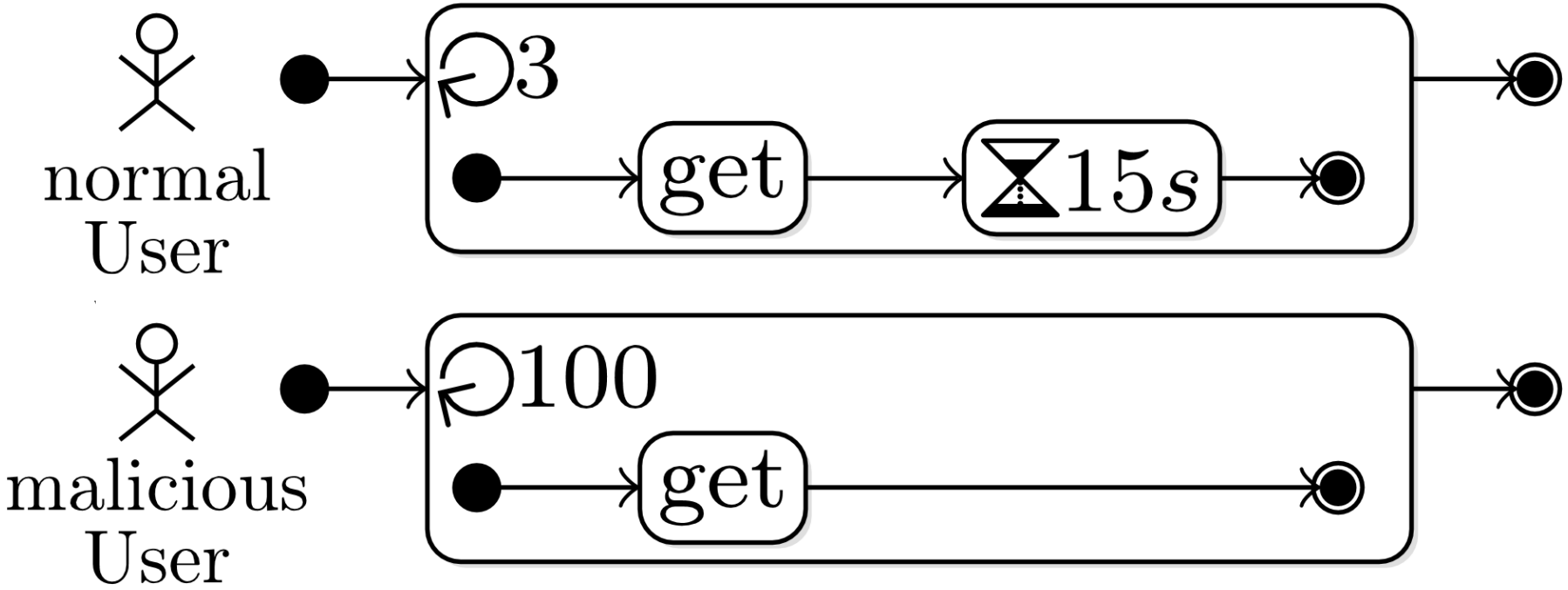
- **Argument:**

  Performance models need to express state
  for **separating concerns/roles** (in specific cases)

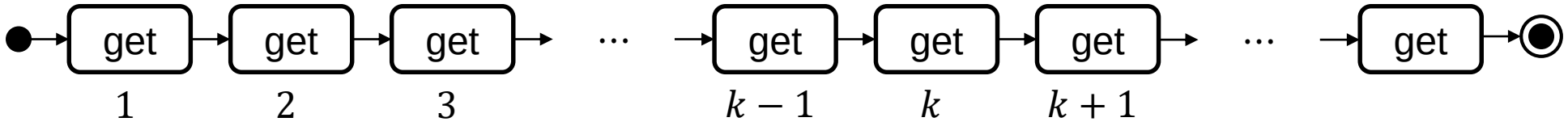
- **Approach (conceptual):**

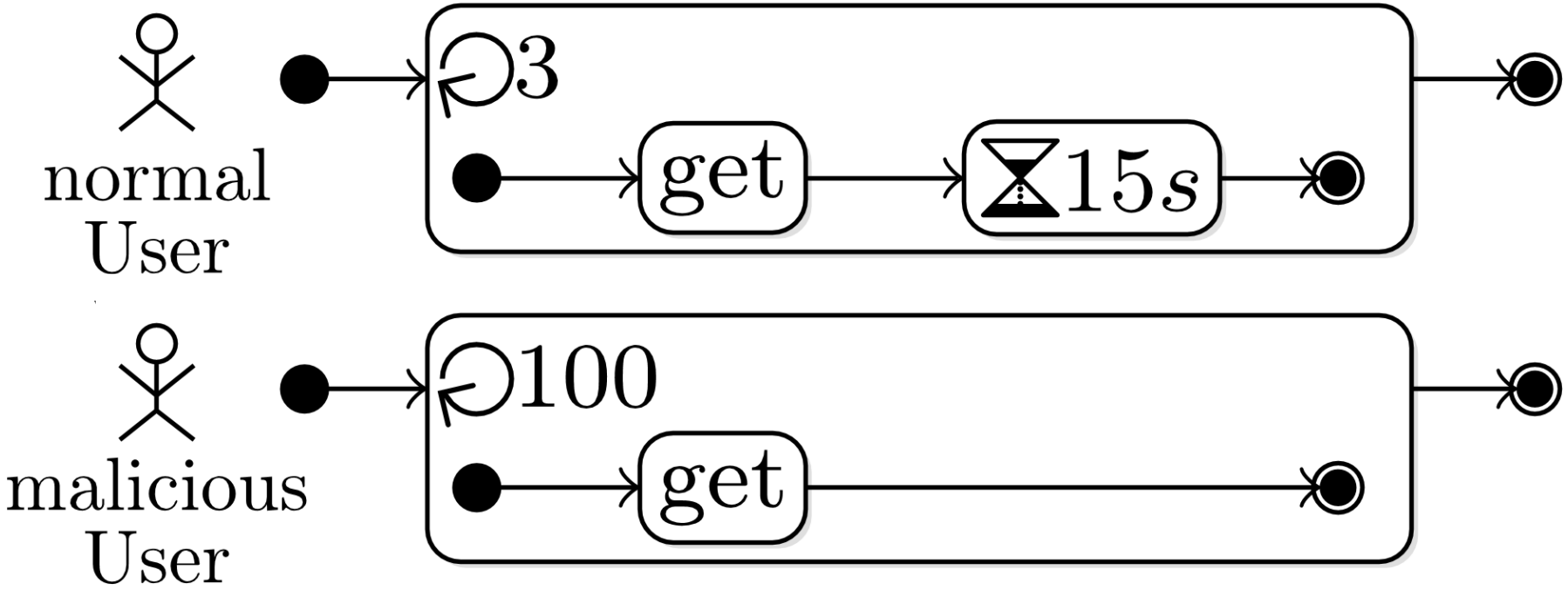
  Extension: stochastic expressions that preserve separation.
  Reduction to ‘stateless’ model.


System Architect
optional
IService
IDS
Service
Component Developer
[deep package inspection]
«CPU demand»
d
IDS.get
[else]
«call»
IService.get
Domain Expert
normal User
3
get
15s
malicious User
100
get

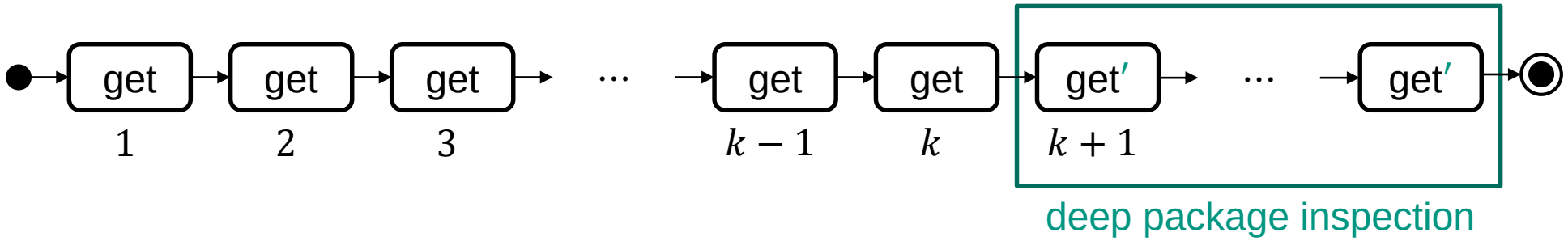


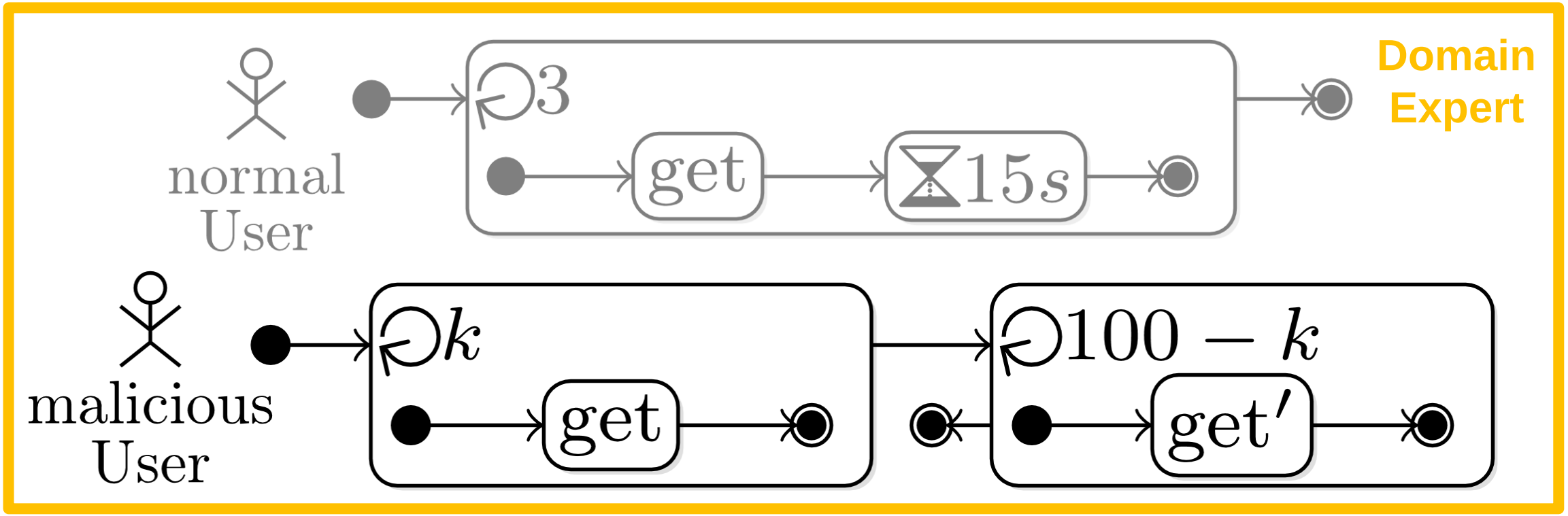
- Desired behavior: after k requests by a user in a session, deep package inspection for all subsequent requests



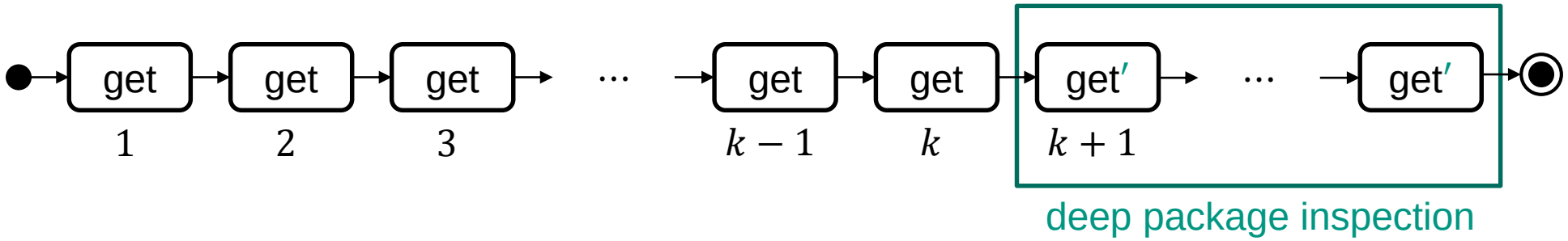


- Desired behavior: after k requests by a user in a session, deep package inspection for all subsequent requests

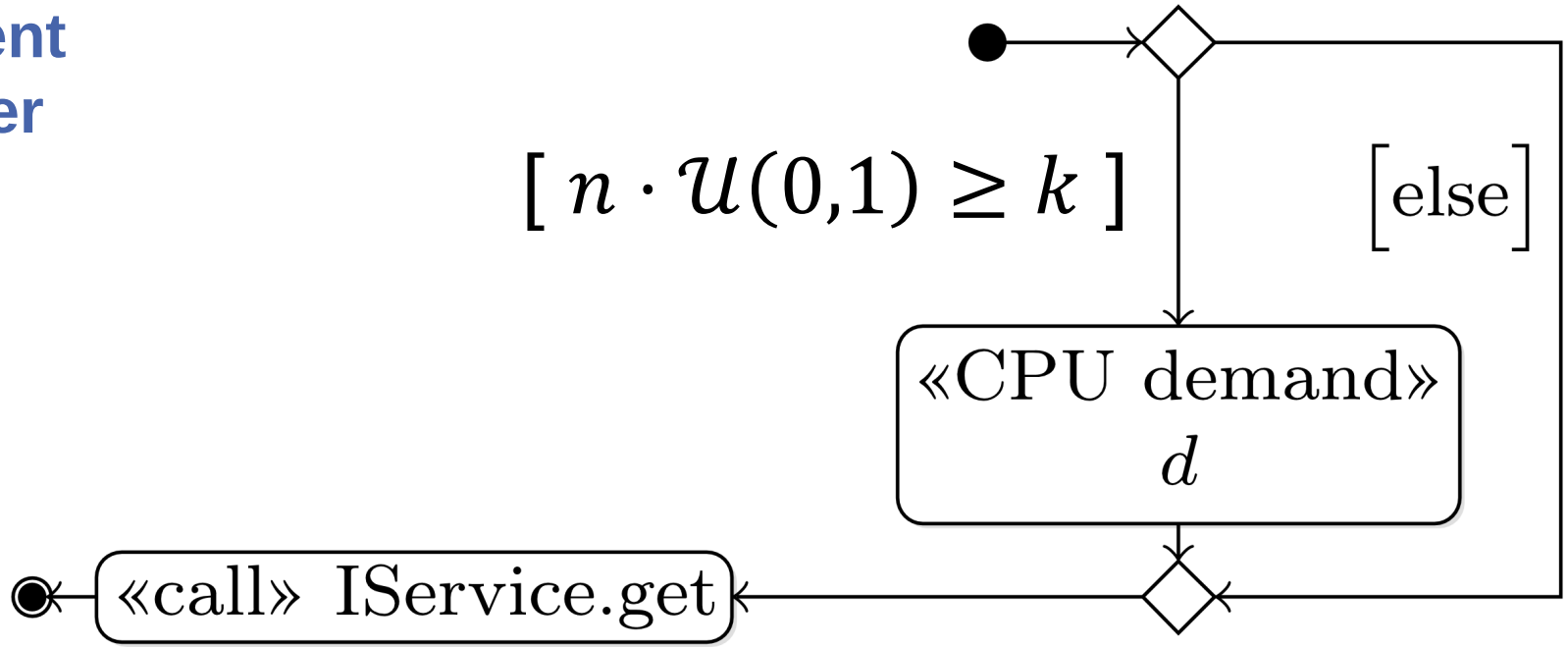


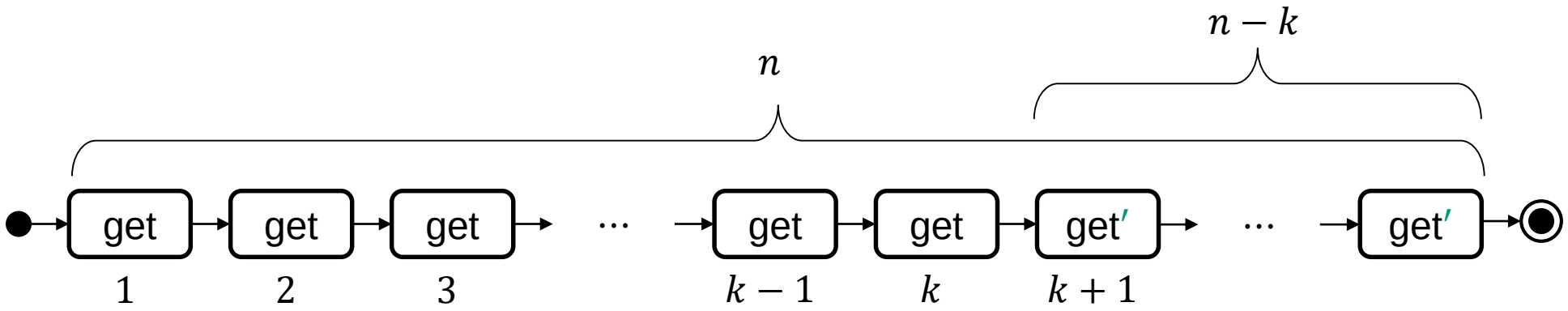


- Desired behavior: after k requests by a user in a session, deep package inspection for all subsequent requests

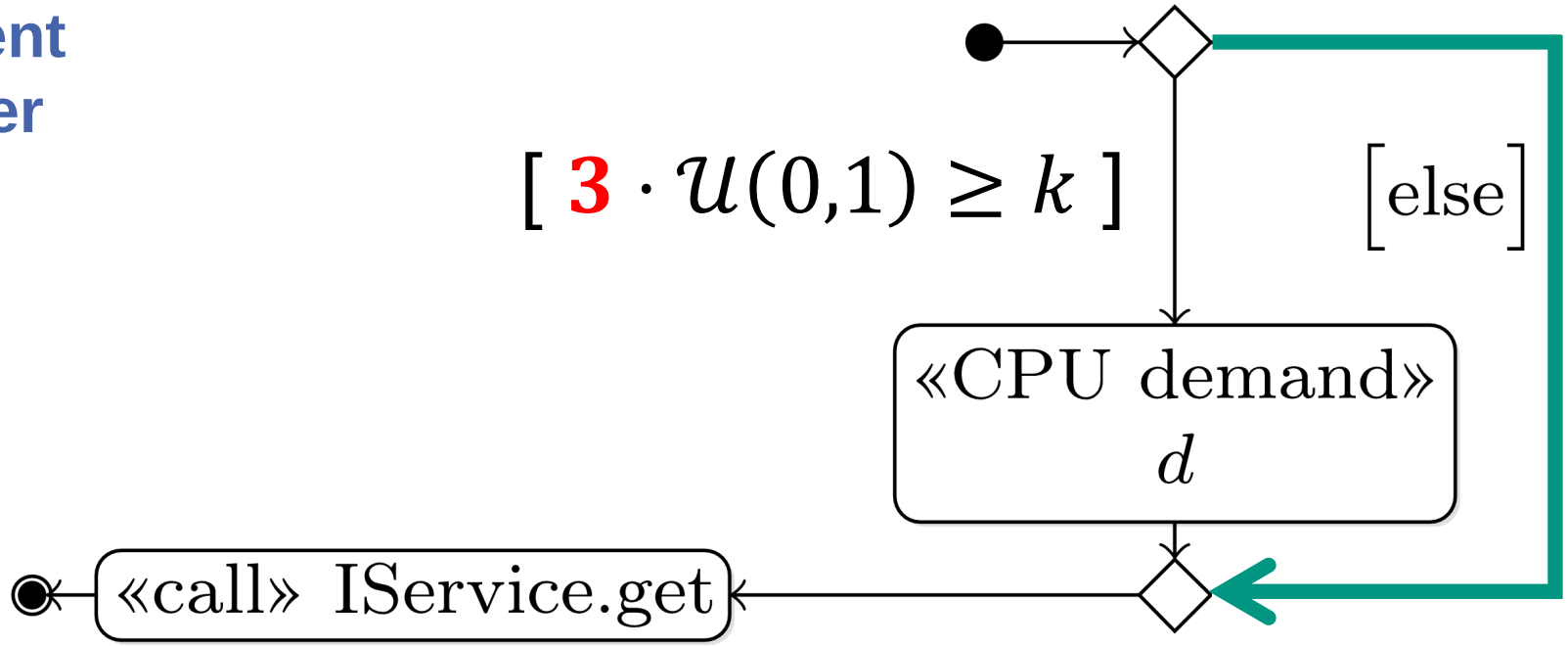


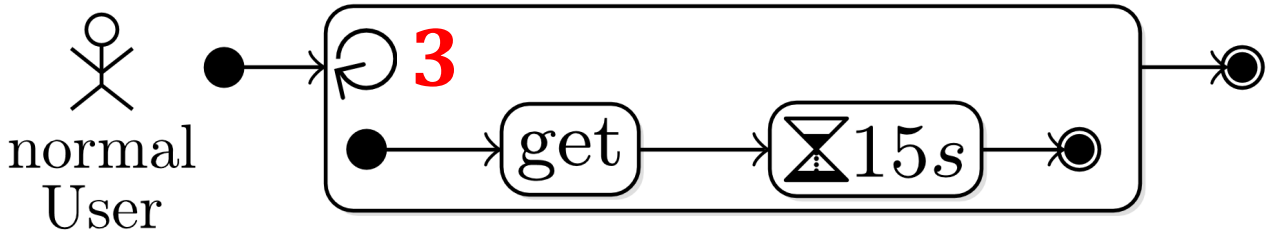
Component Developer



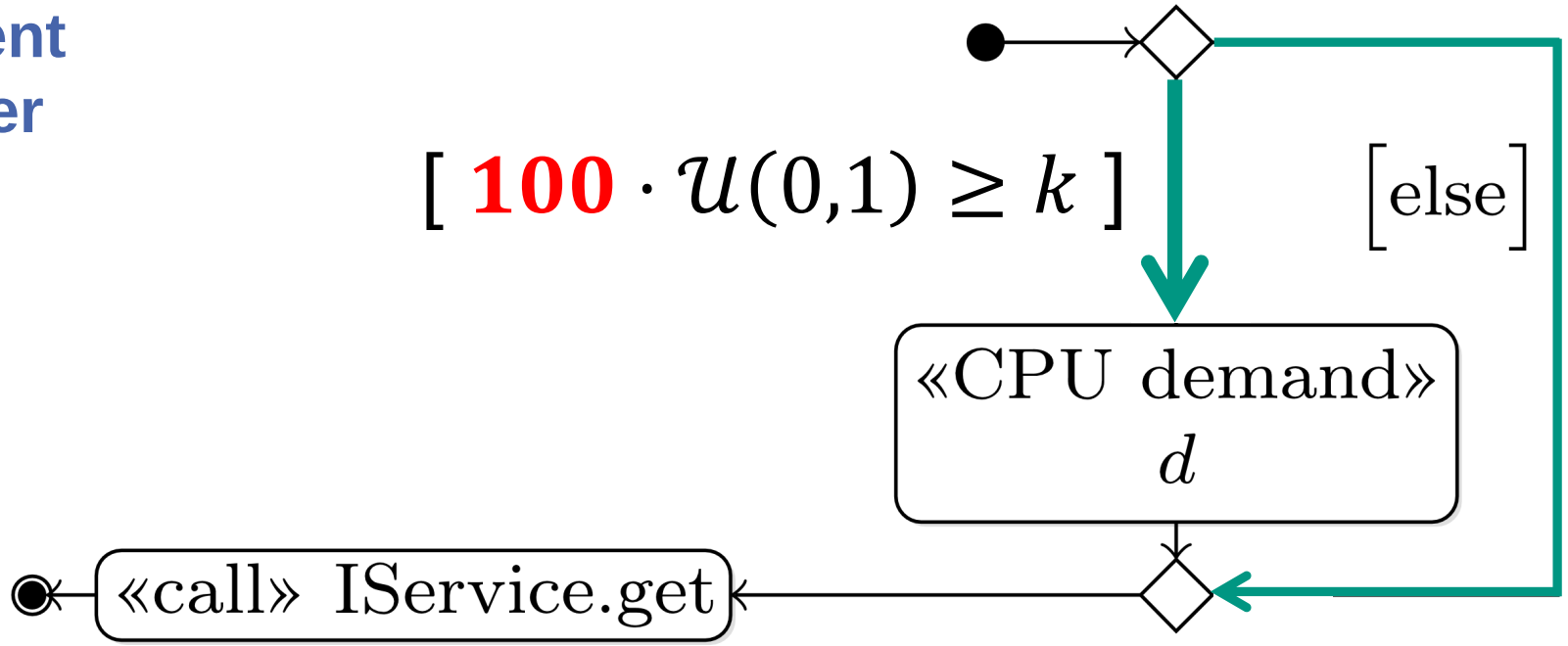


Component Developer

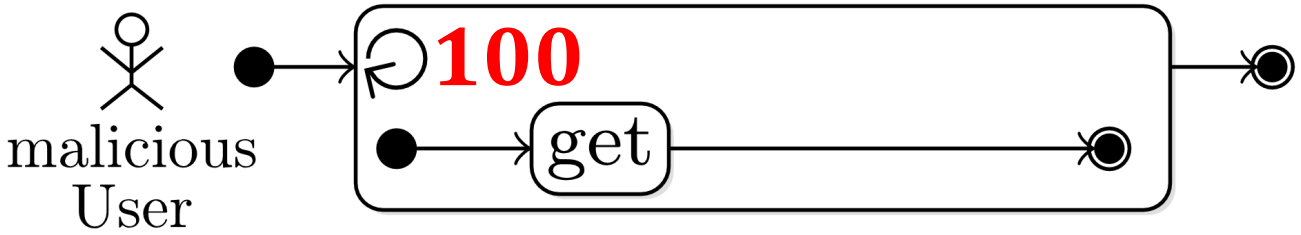




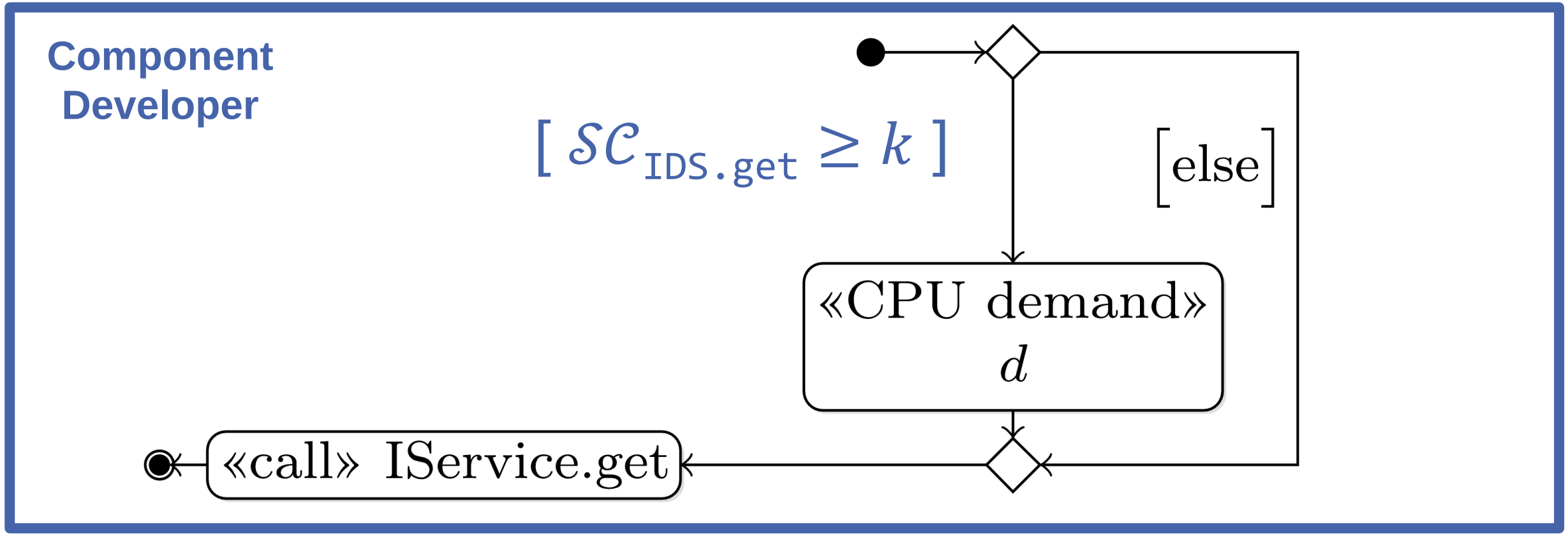
Component Developer



Domain Expert

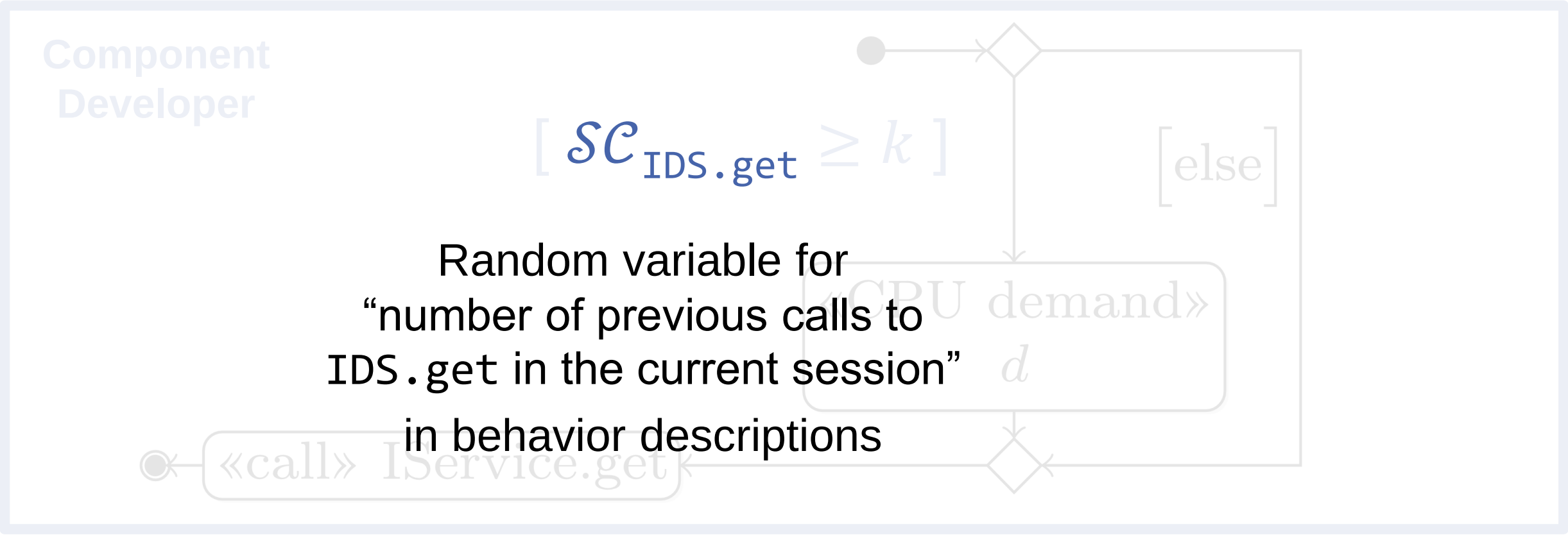


Approach: Session Count Expressions



- Express number of previous calls to a method in the current session
- Transformation to approximation

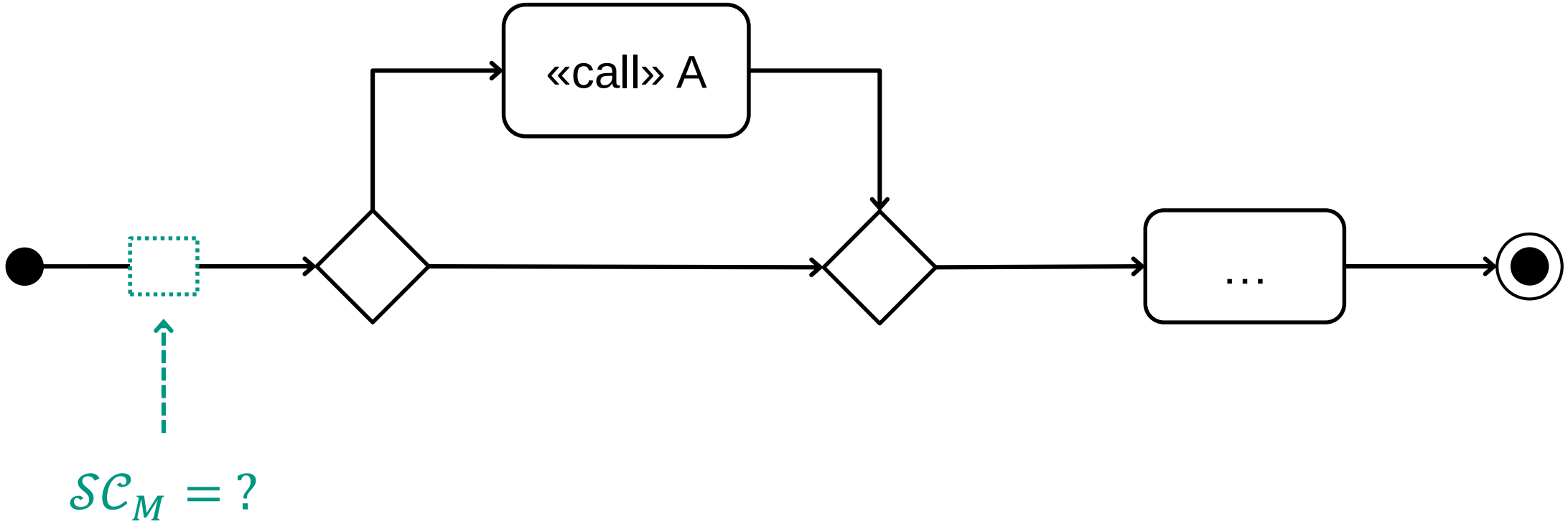
Approach: Session Count Expressions



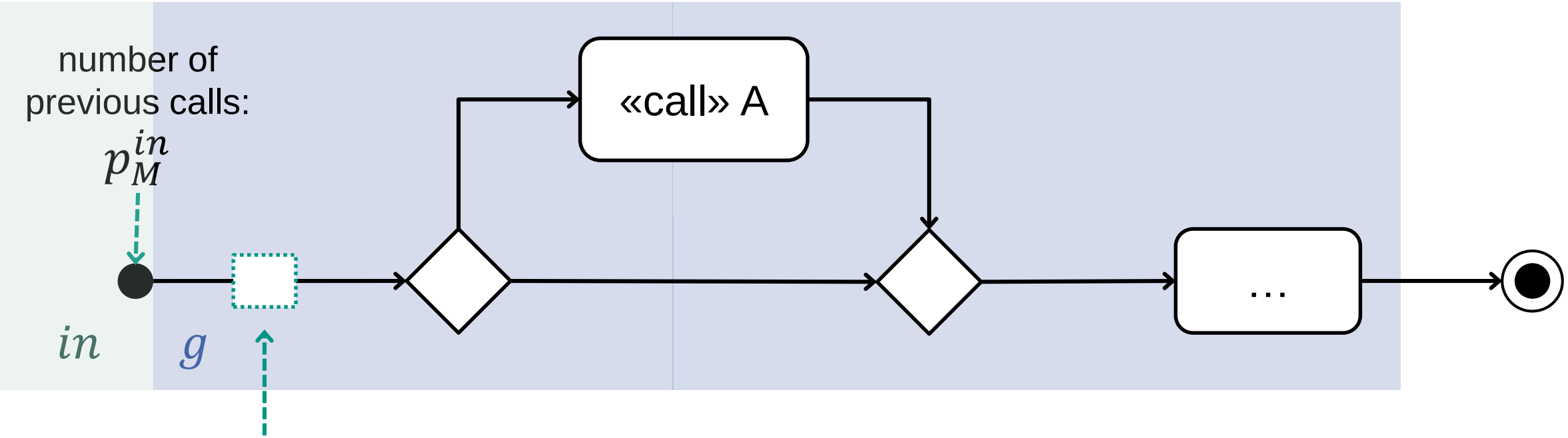
Random variable for
"number of previous calls to
`IDS.get` in the current session"
in behavior descriptions

- Express number of previous calls to a method in the current session
- Transformation to approximation

Approach: Session Count Expressions

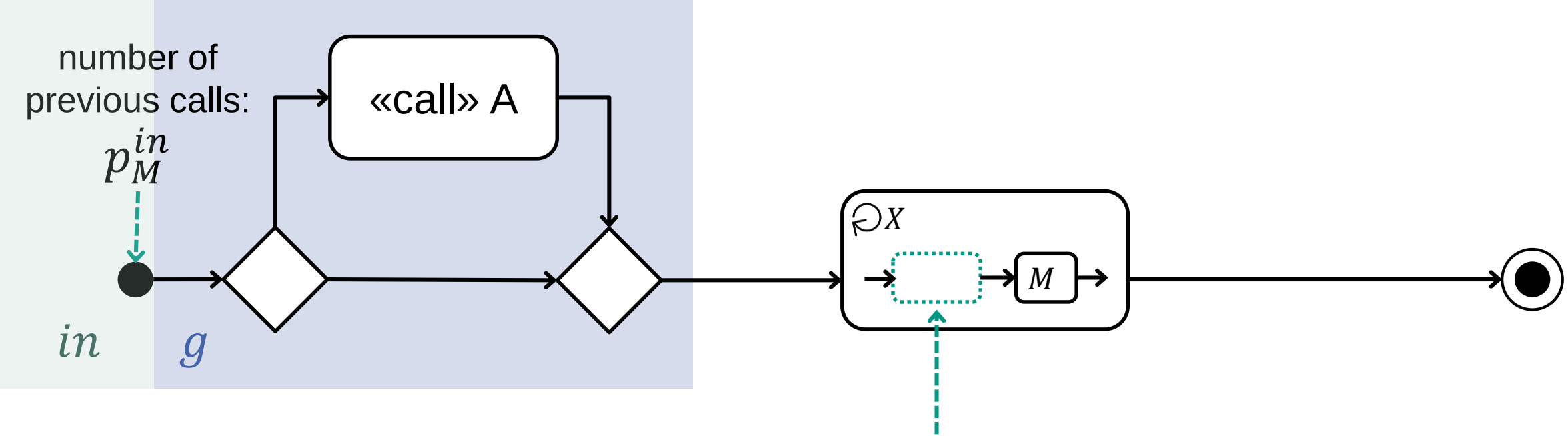


Approach: Session Count Expressions



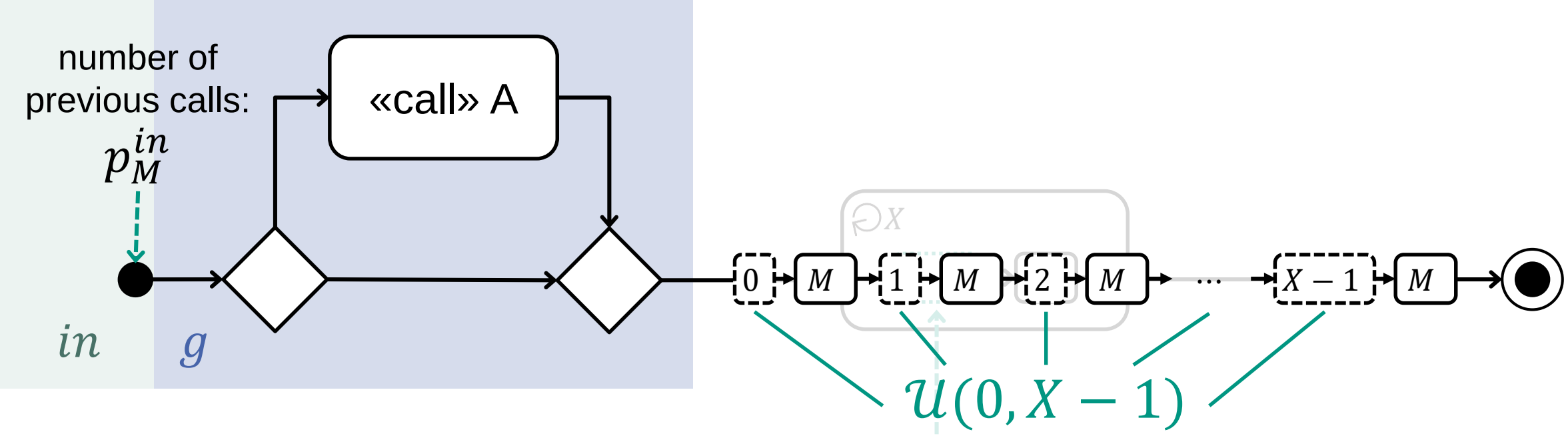
$$\mathcal{SC}_M = p_M^{in} + c_M^g$$

Approach: Session Count Expressions



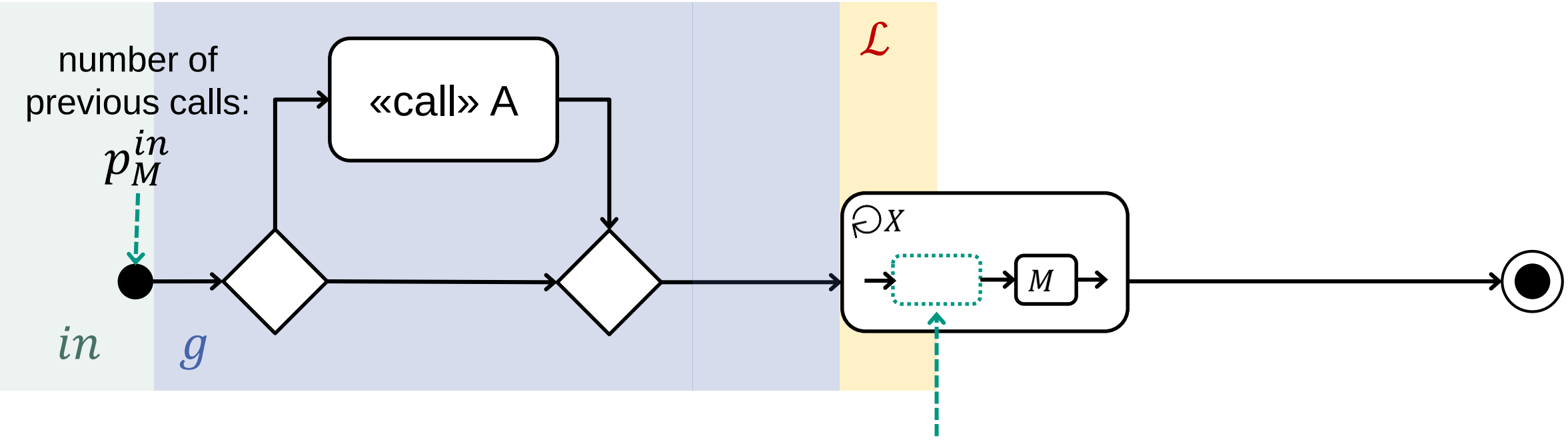
$$\mathcal{SC}_M = p_M^{in} + c_M^g$$

Approach: Session Count Expressions



$$\mathcal{SC}_M = p_M^{in} + c_M^g$$

Approach: Session Count Expressions



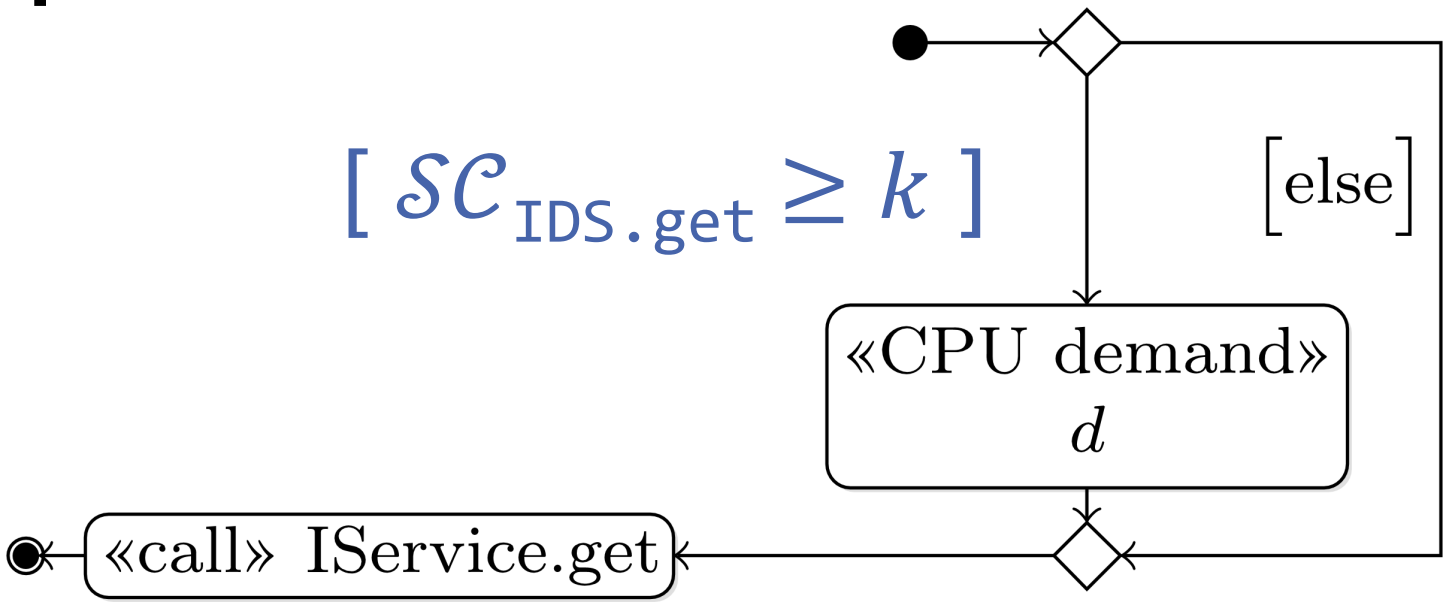
$$\mathcal{SC}_M = p_M^{in} + c_M^g + \mathcal{L}$$

Approach: Session Count Expressions

Transformation sketch:

1. for each used $\mathcal{SC}_M$:
   add a parameter p_M^{in} to *all* signatures of *all* methods
2. replace each reference to $\mathcal{SC}_M$ by $\mathcal{SC}_M = p_M^{in} + c_M^g + \mathcal{L}$
   (g: subgraph of all preceding actions)
   loop approximation $\mathcal{L} = 0$, except for references to $\mathcal{SC}_m$ inside loops with calls to M. Inside a loop (X iterations): $\mathcal{L} = \mathcal{U}(0, X - 1)$
3. for each service call:
   pass $\mathcal{SC}_M = p_M^{in} + c_M^g + \mathcal{L}$ as the parameter p_M^{in}.

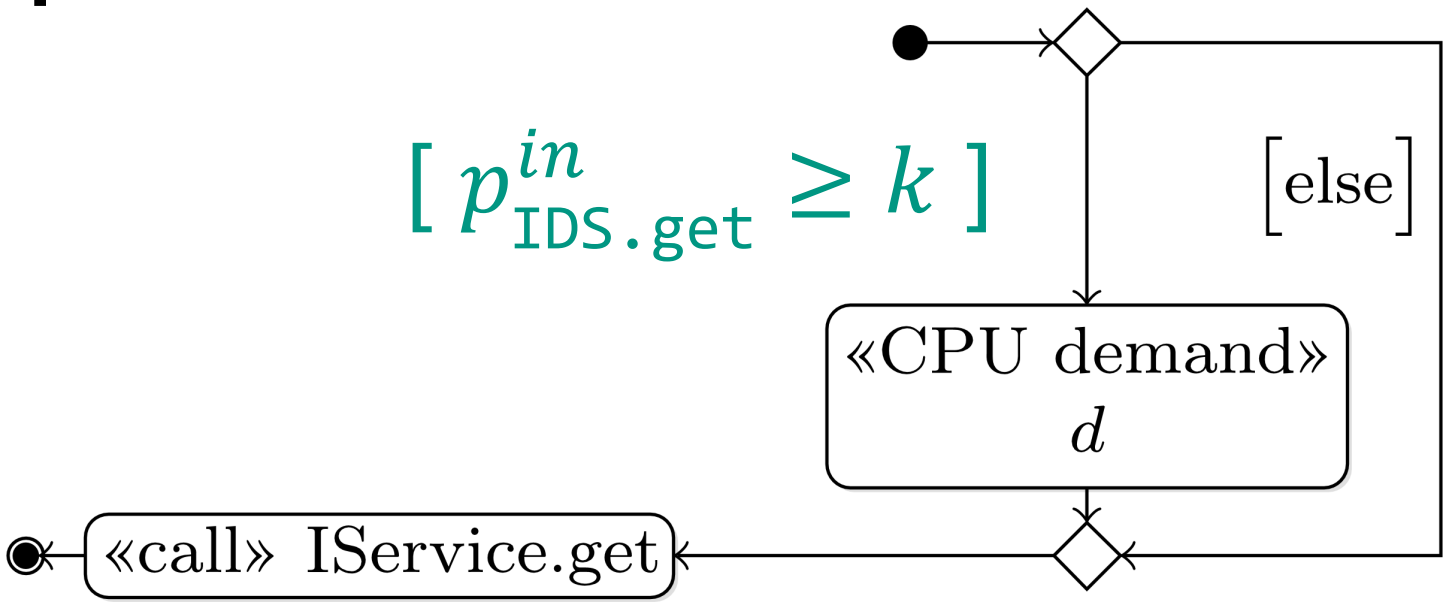
Example application



1. for each used $\mathcal{SC}_M$:
   add a parameter p_M^{in} to *all* signatures of *all* methods

⇒ add the parameter $p_{\texttt{IDS.get}}^{in}$ to the signature of `IService.get`

Example application

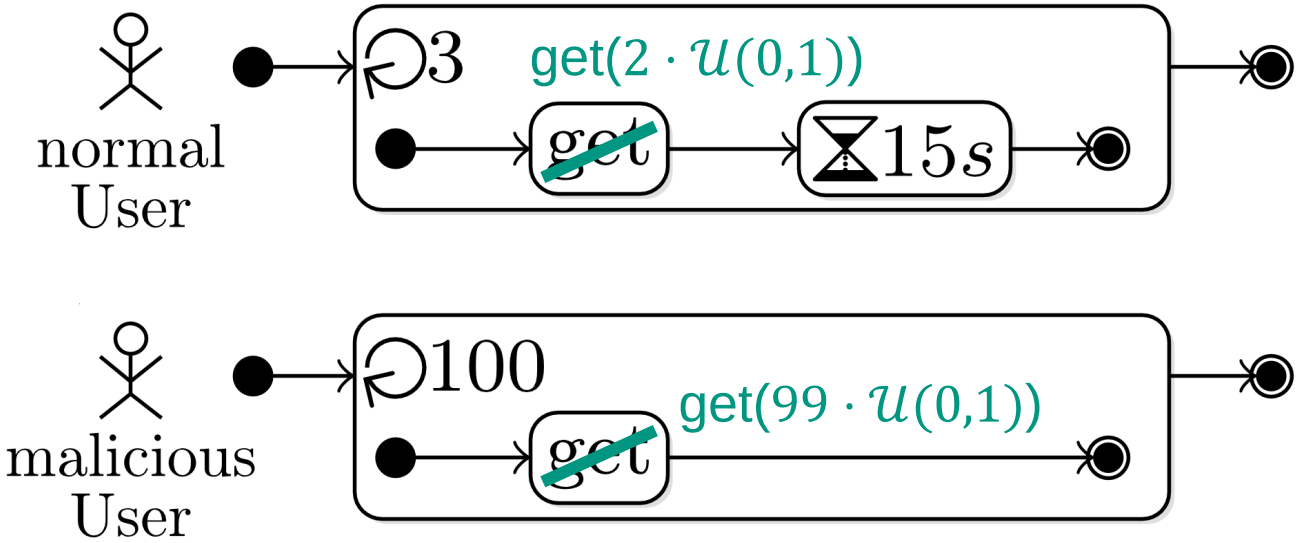


2. replace each reference to $\mathcal{SC}_M$ by $\mathcal{SC}_M = p^{in}_M + c^g_M + \mathcal{L}$
   (g: subgraph of all preceding actions)

⇒ no other calls to `get` ($c^g_M = 0$), not inside loop ($\mathcal{L} = 0$), therefore

$$\mathcal{SC}_{\texttt{IDS.get}} = p^{in}_{\texttt{IDS.get}}$$

Example application



1. for each service call:
   pass $\mathcal{SC}_M = p_M^{in} + c_M^g + \mathcal{L}$ as the parameter p_M^{in}.

$$p_M^{in} = 0,\ c^g_{\texttt{IDS.get}} = 0,\ \mathcal{L} = 2 \cdot \mathcal{U}(0,1),\ \text{resp. } \mathcal{L} = 99 \cdot \mathcal{U}(0,1)$$

Stateful Palladio

- Happe et al. (2013): Stateful Palladio
- allow the definition and manipulation of state of:
  - components,
  - systems,
  - users,
  - session
- Simulator is extended to support stateful models

L. Happe, B. Buhnova, and R. Reussner. “Stateful component-based performance models”. In: Software & Systems Modeling 13.4 (2013), pp. 1319–1343.

Conclusion

- Specific modeling scenarios break separation of concerns between roles for performance modeling
- Session Count Expressions help in specific cases (number of calls in a session)

- Context: Method for modeling behavior which depends on *sets* of requests while preserving separation of concerns (≠ call & return)
- Future work:
  - Implementation and evaluation, approximation “good enough?”
  - Other state approximations (component, system, user), if possible