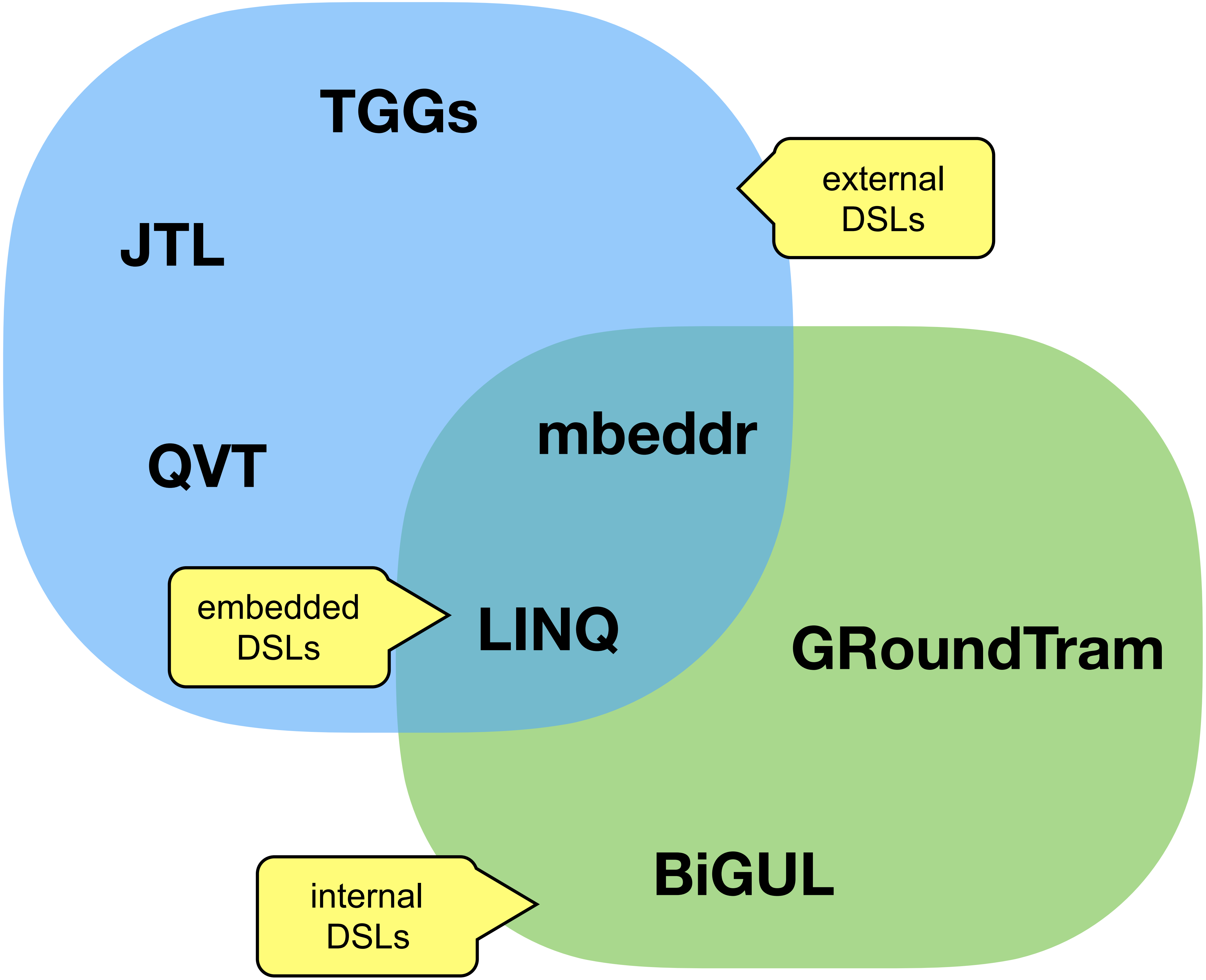
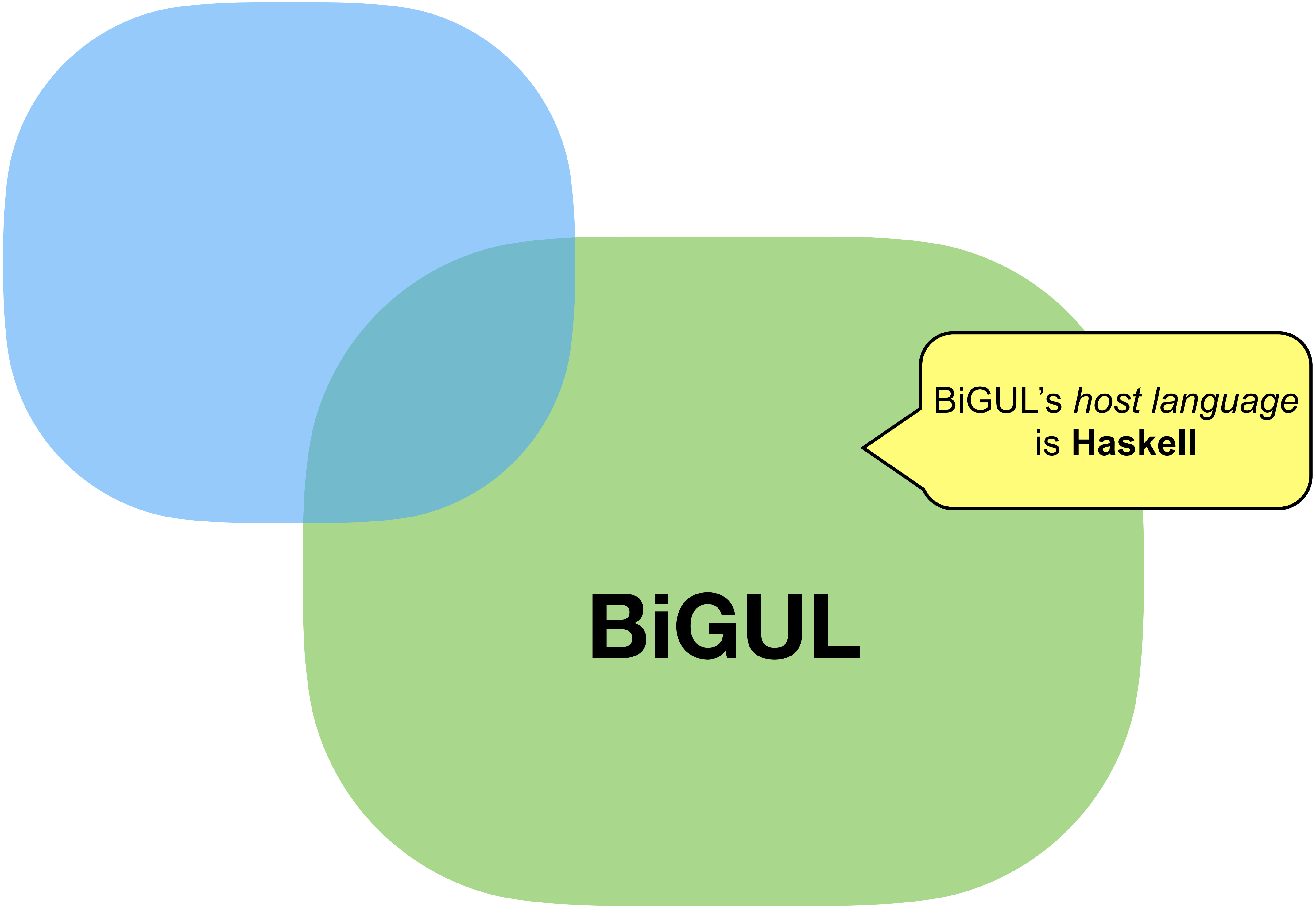
Towards a Visual Editor for Lens Combinators

Tony Anjorin (Paderborn University, Germany)
Josh Ko (National Institute of Informatics, Japan)

International Workshop on Bidirectional Transformations (BX)
10 April 2018, Nice, France


TGGs
external DSLs
JTL
mbeddr
QVT
embedded DSLs
LINQ
GRoundTram
internal DSLs
BiGUL

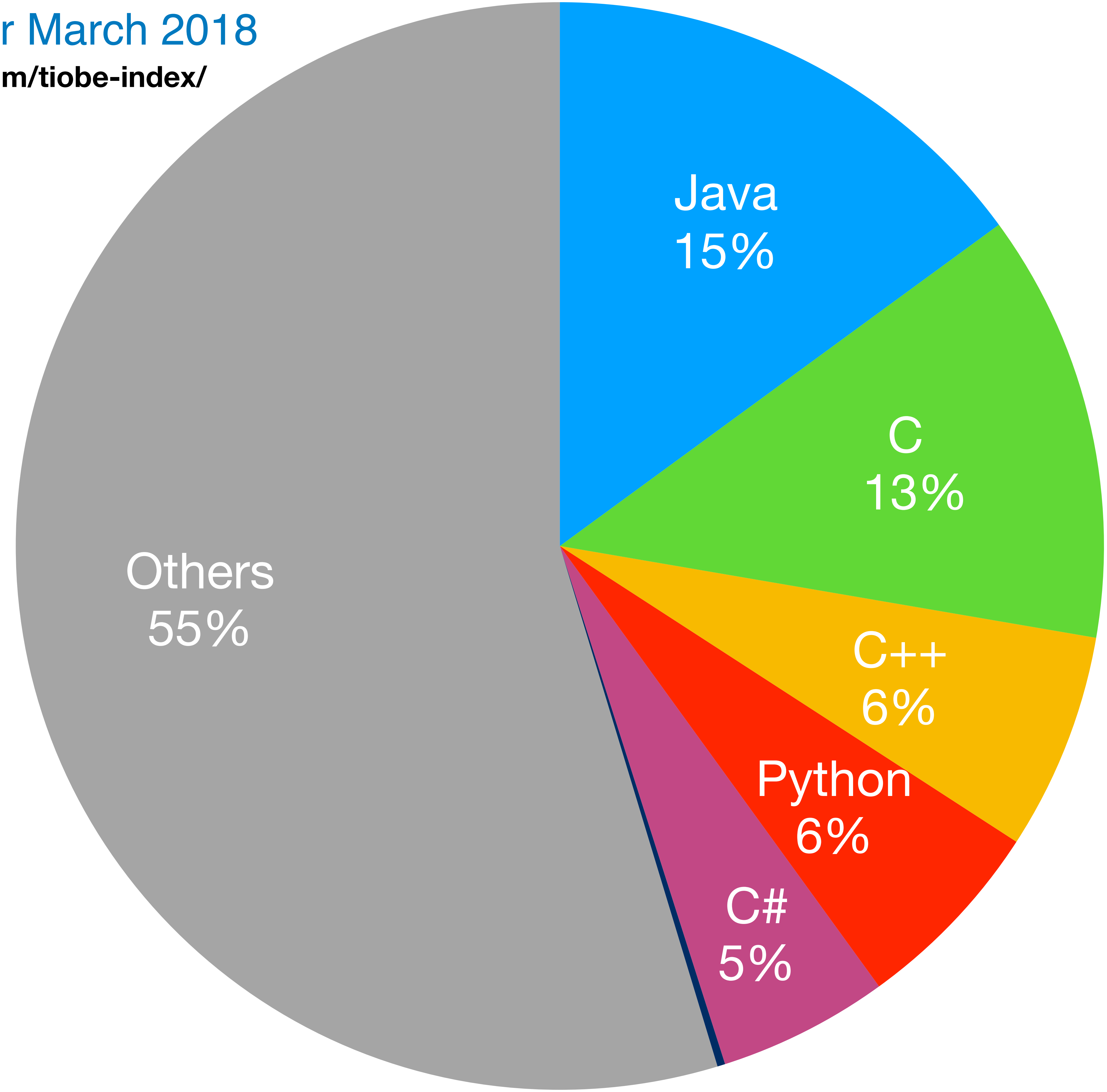


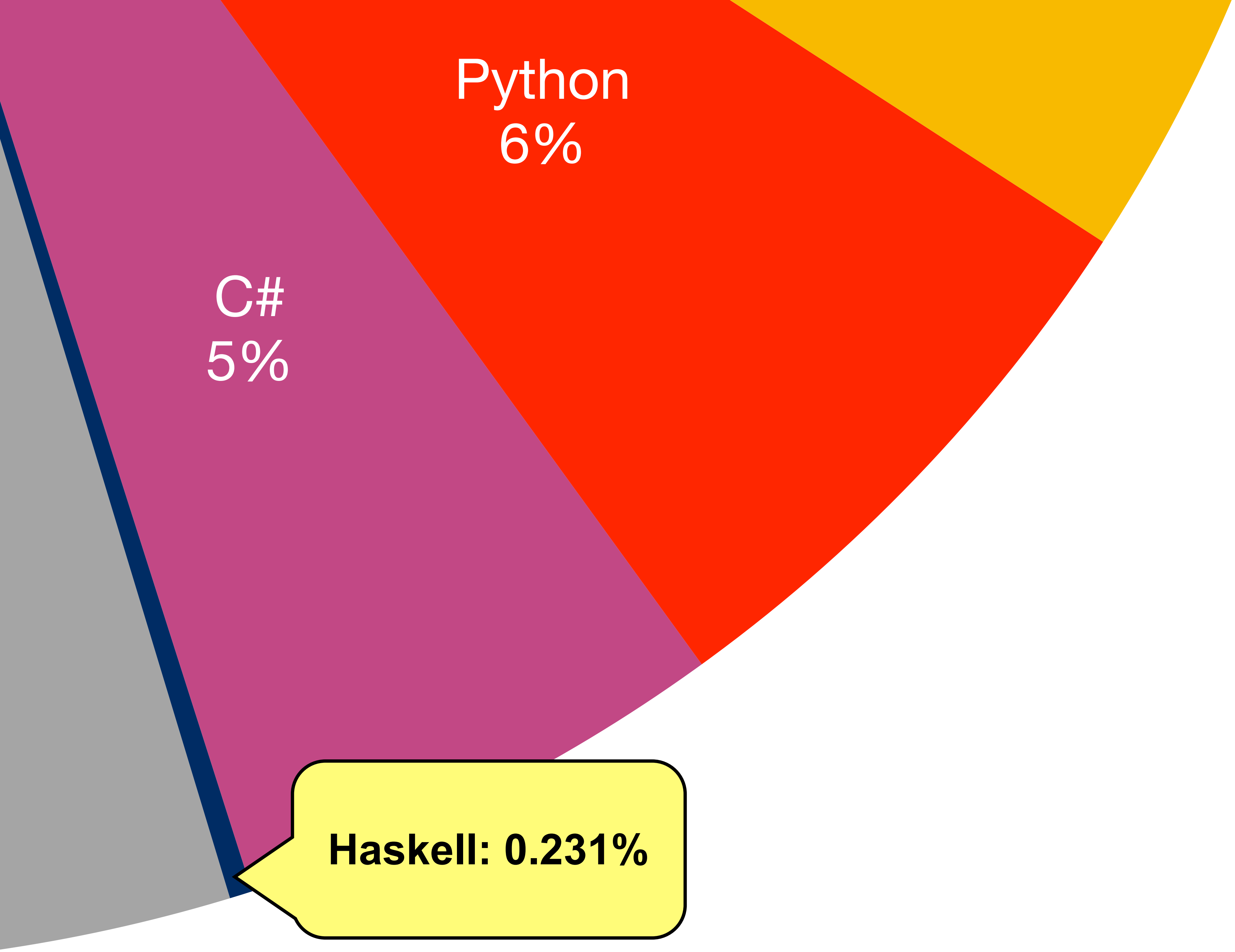
BiGUL

BiGUL's *host language*
is **Haskell**

TIOBE Index for March 2018

https://www.tiobe.com/tiobe-index/



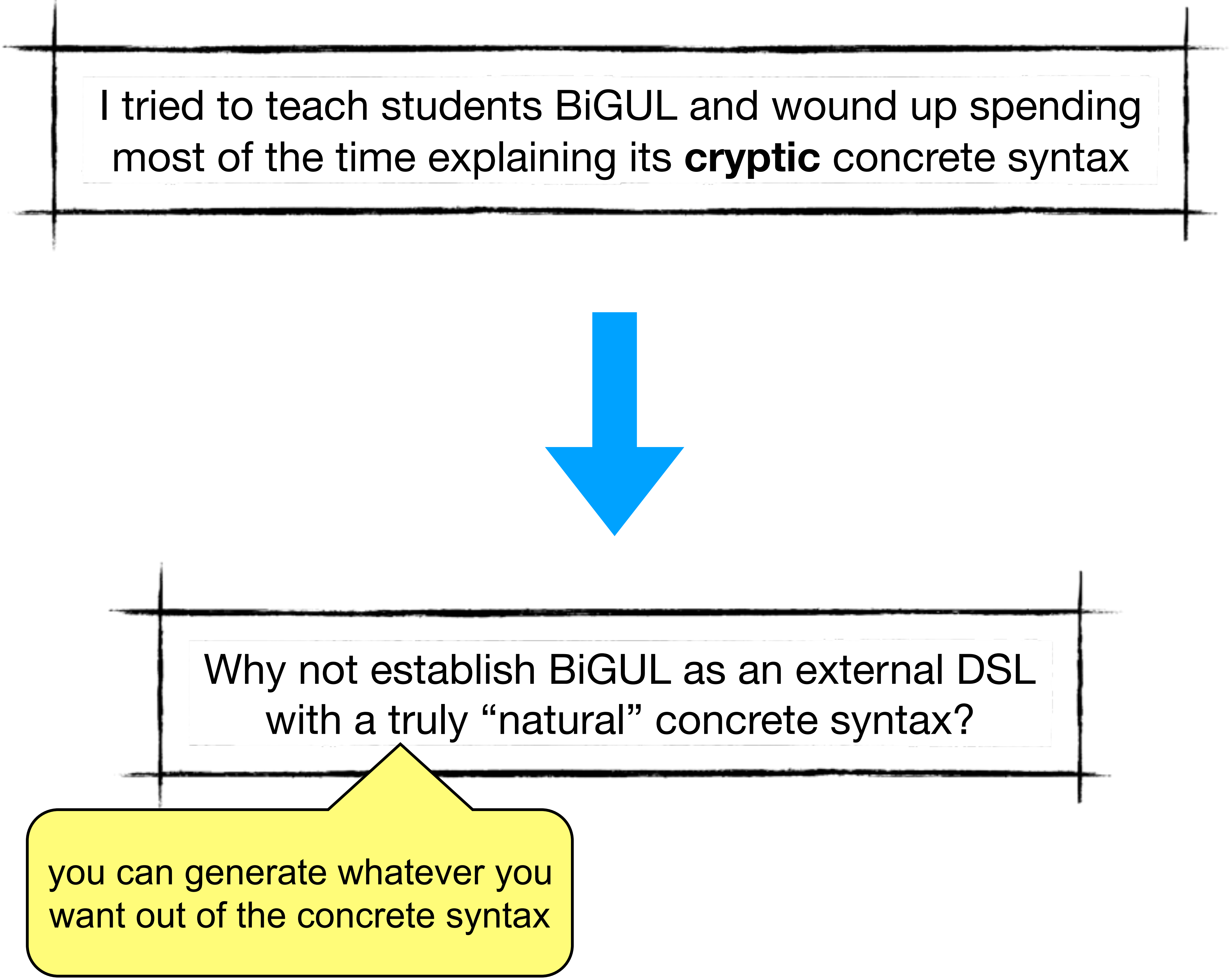

Python
6%
C#
5%
Haskell: 0.231%

My Conjecture

Haskell is a great language with a concise,
elegant concrete syntax, but …

… it is unfamiliar to **most** programmers
and is thus hard to learn and read

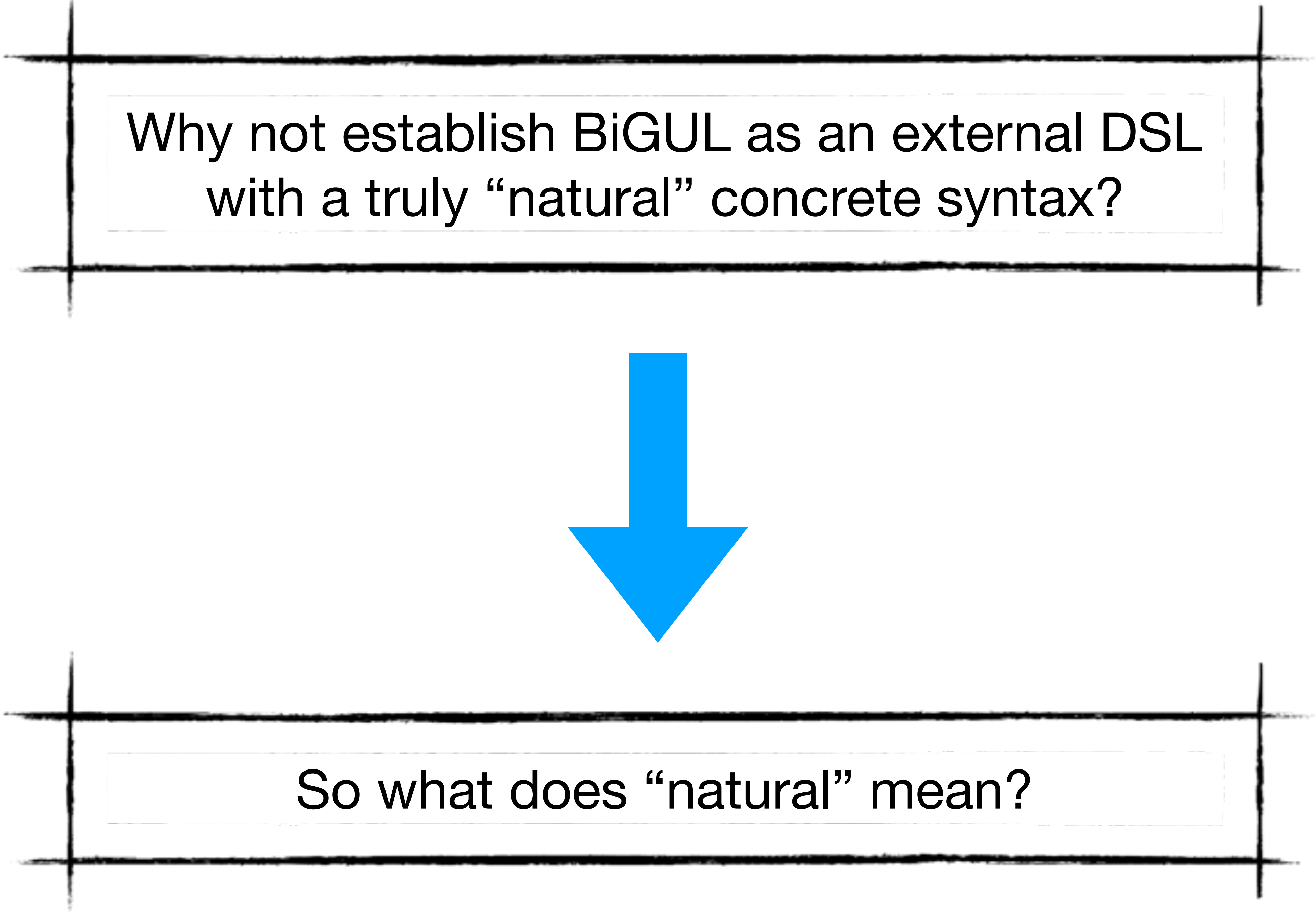
I tried to teach students BiGUL and wound up spending
most of the time explaining its **cryptic** concrete syntax



I tried to teach students BiGUL and wound up spending most of the time explaining its **cryptic** concrete syntax

Why not establish BiGUL as an external DSL with a truly “natural” concrete syntax?

you can generate whatever you want out of the concrete syntax



Why not establish BiGUL as an external DSL with a truly “natural” concrete syntax?

So what does “natural” mean?

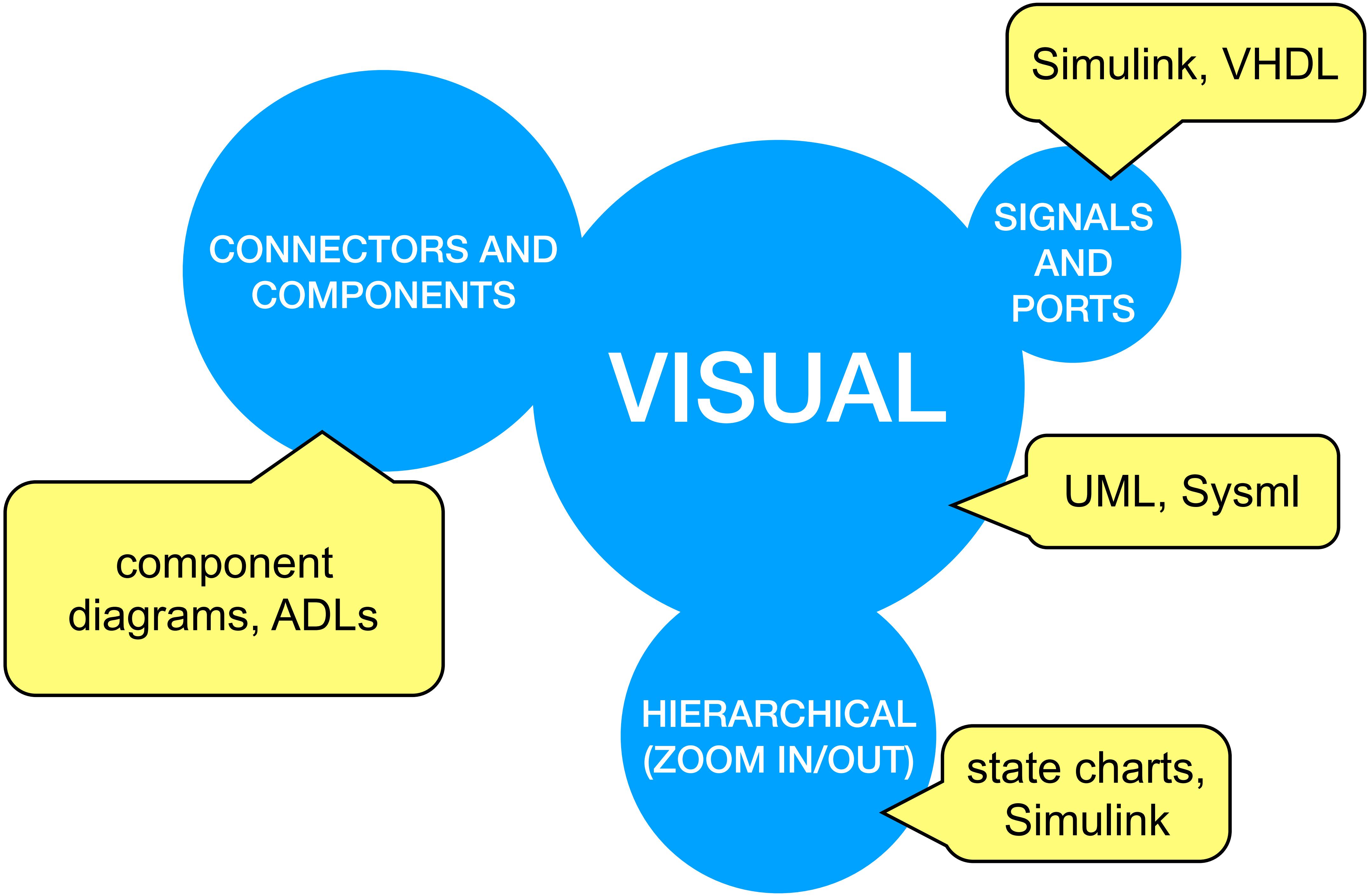
So what does “natural” mean?

My students (and I tend to agree) say:

VISUAL

So what does “natural” mean?

My students (and I tend to agree) say:



VISUAL

CONNECTORS AND COMPONENTS

component diagrams, ADLs

SIGNALS AND PORTS

Simulink, VHDL

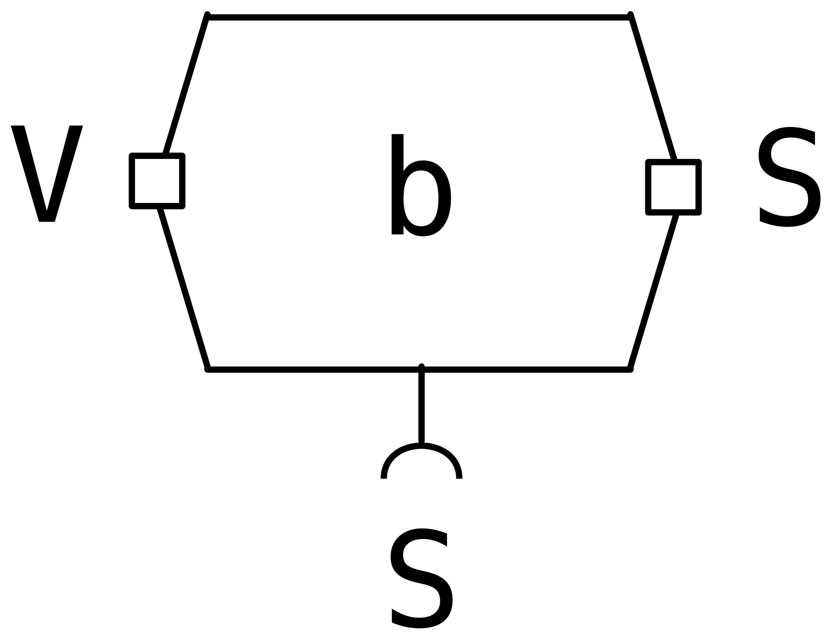
UML, Sysml

HIERARCHICAL (ZOOM IN/OUT)

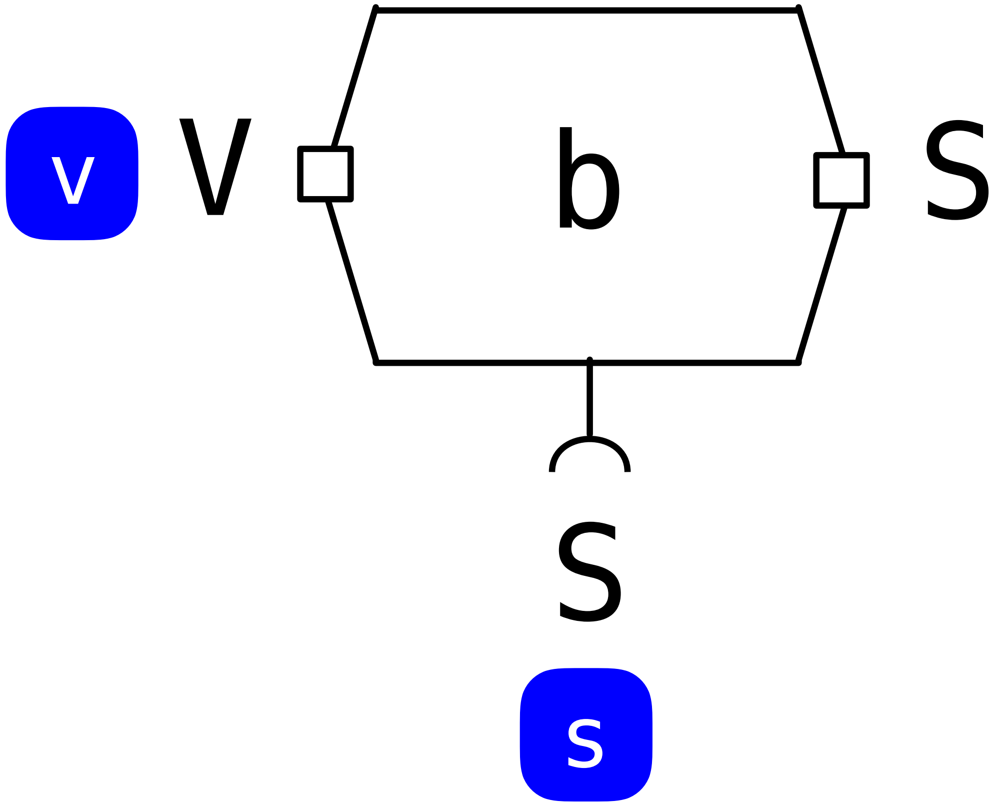
state charts, Simulink

VISUAL BiGUL

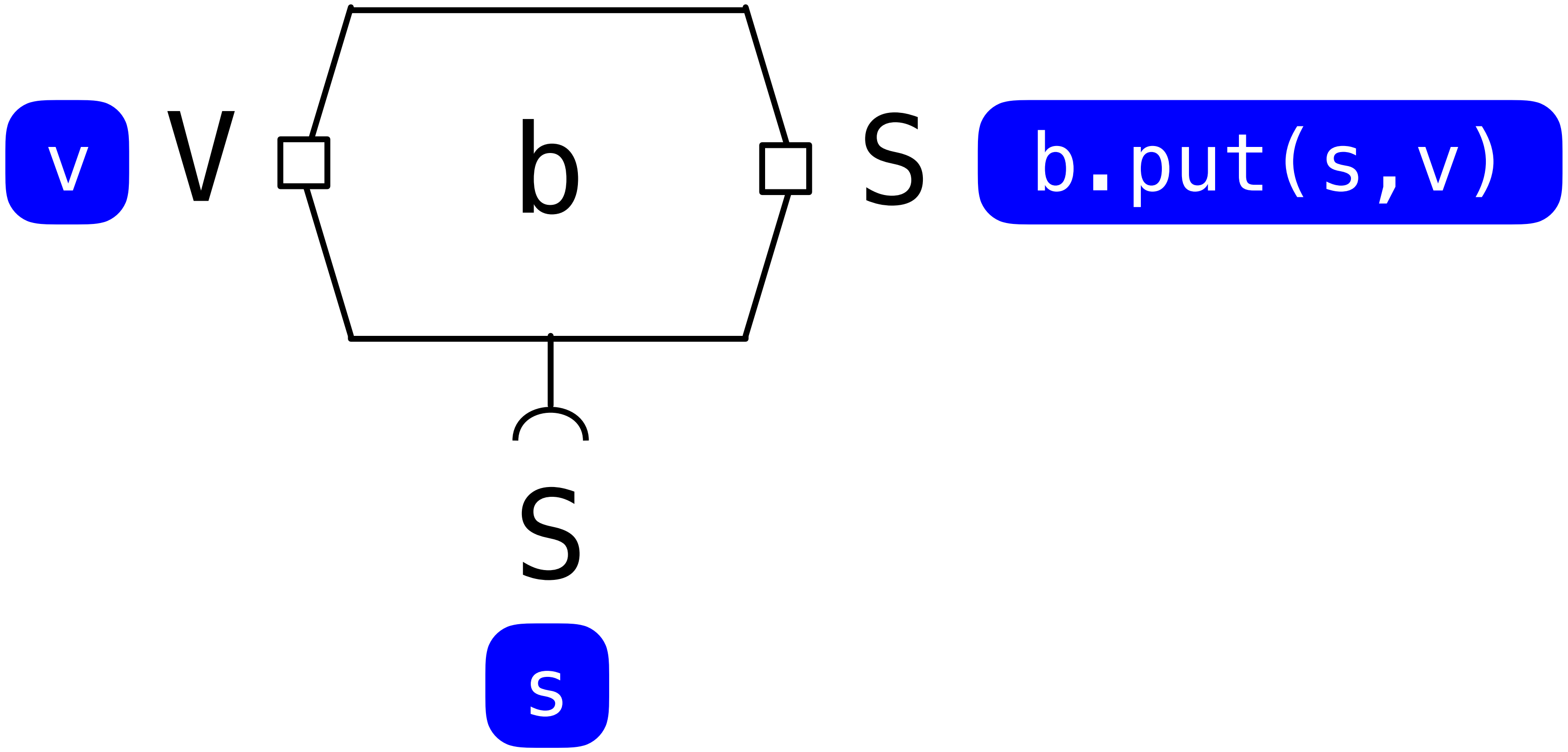
Put Block



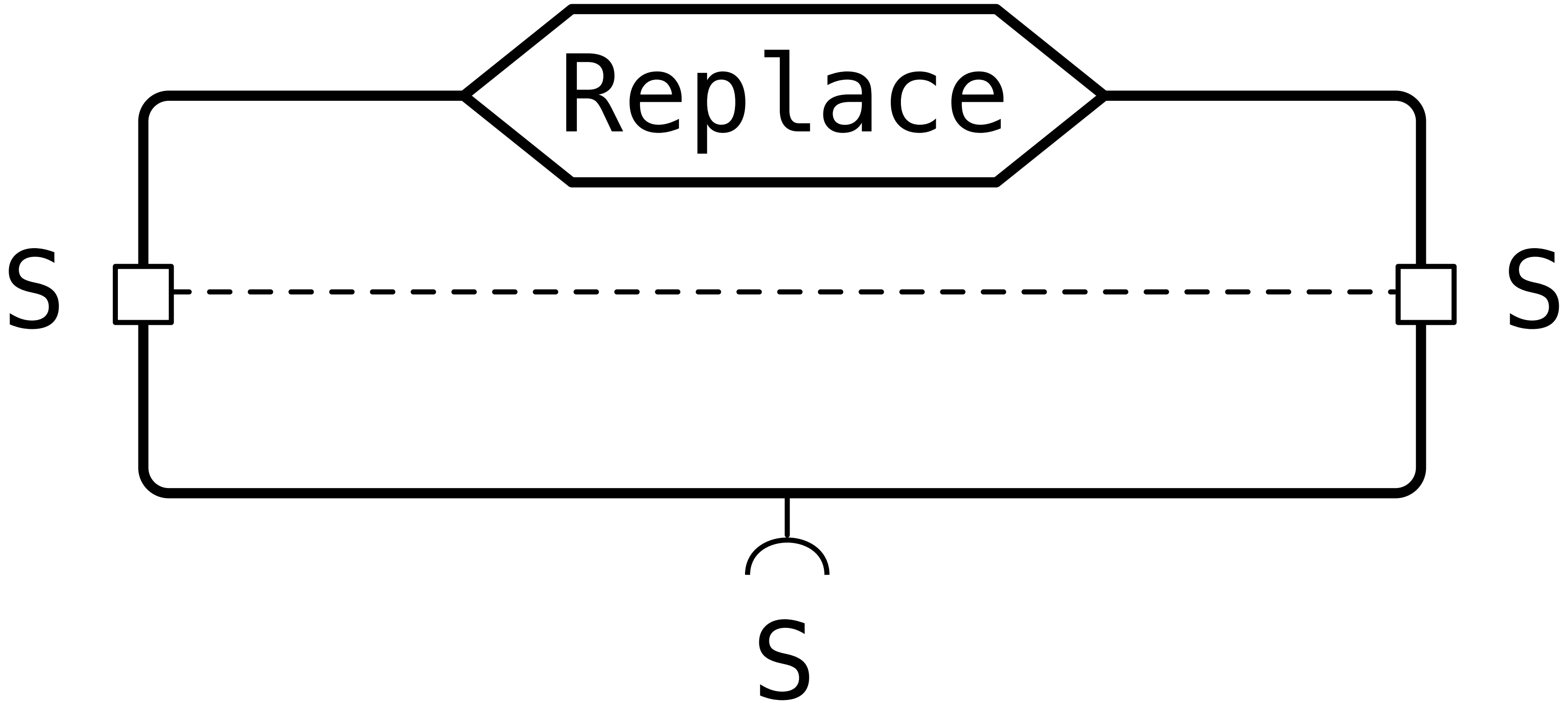
Put Block



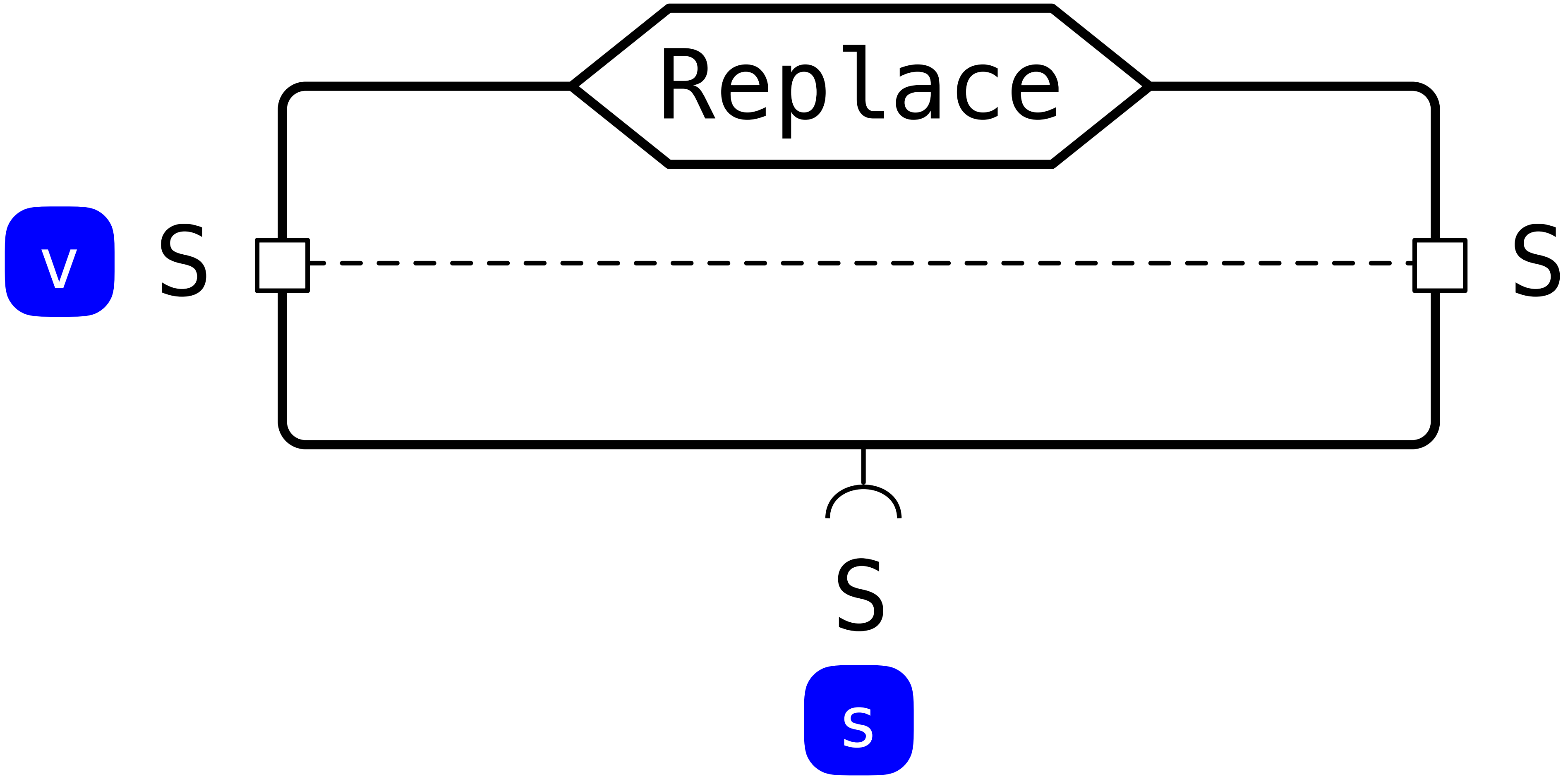
Put Block



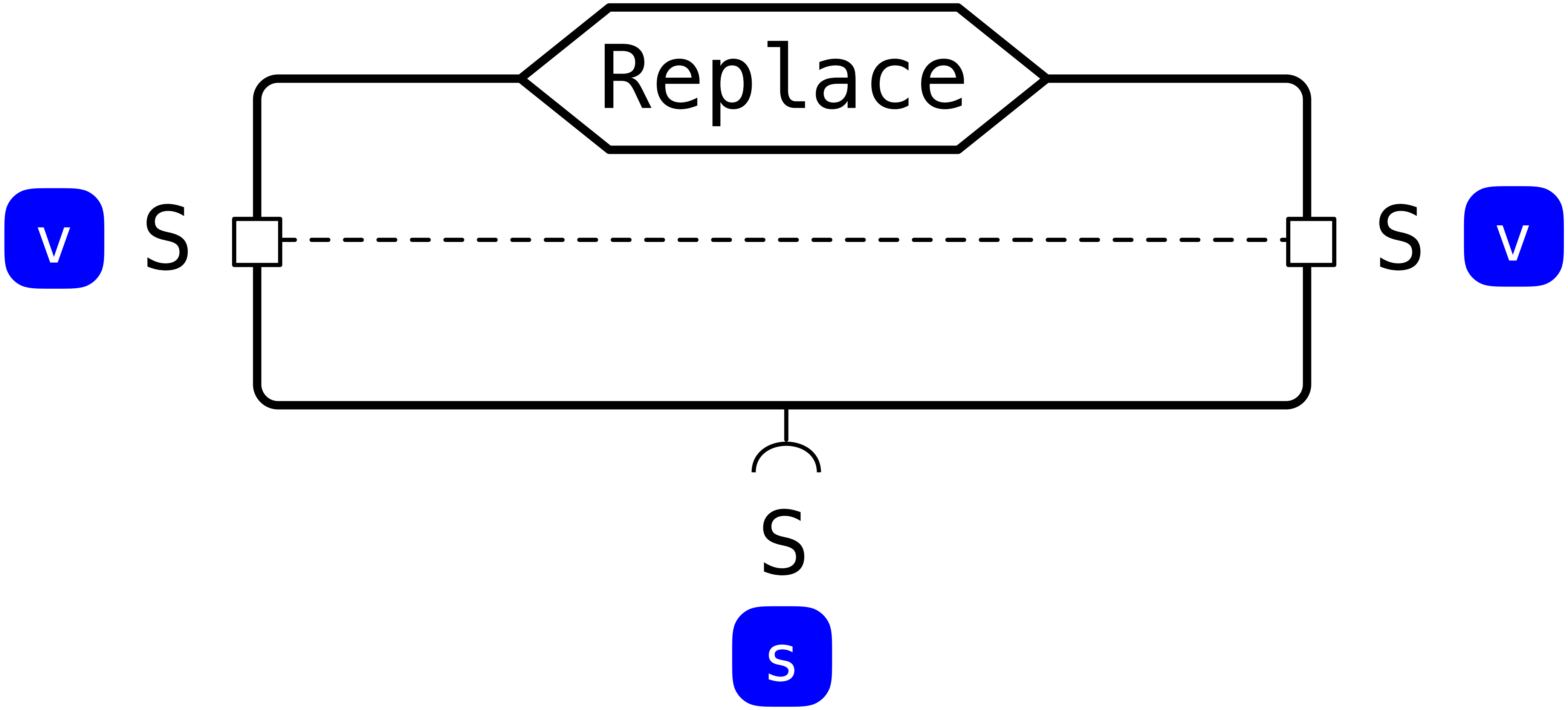
Replace



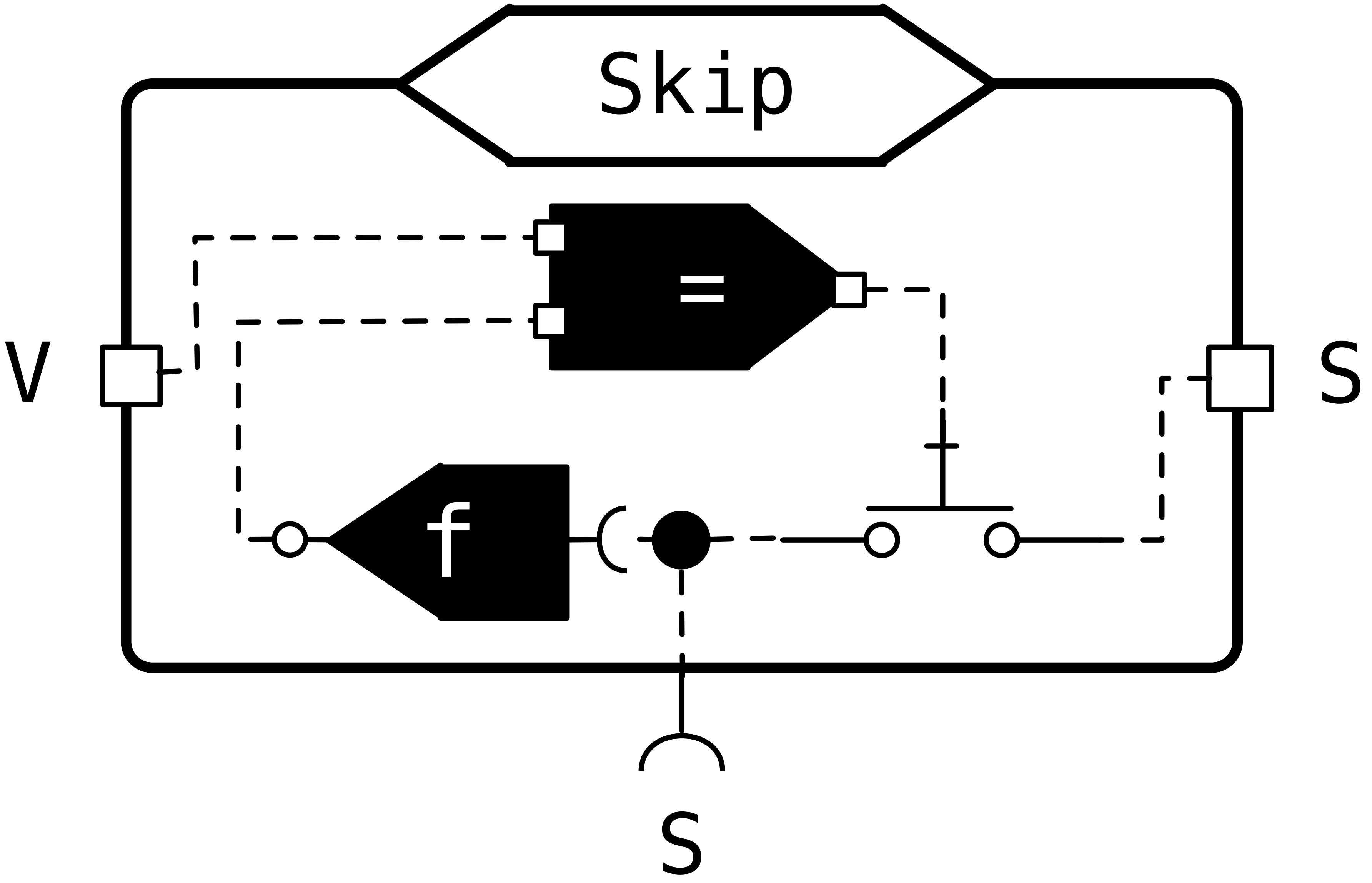
Replace



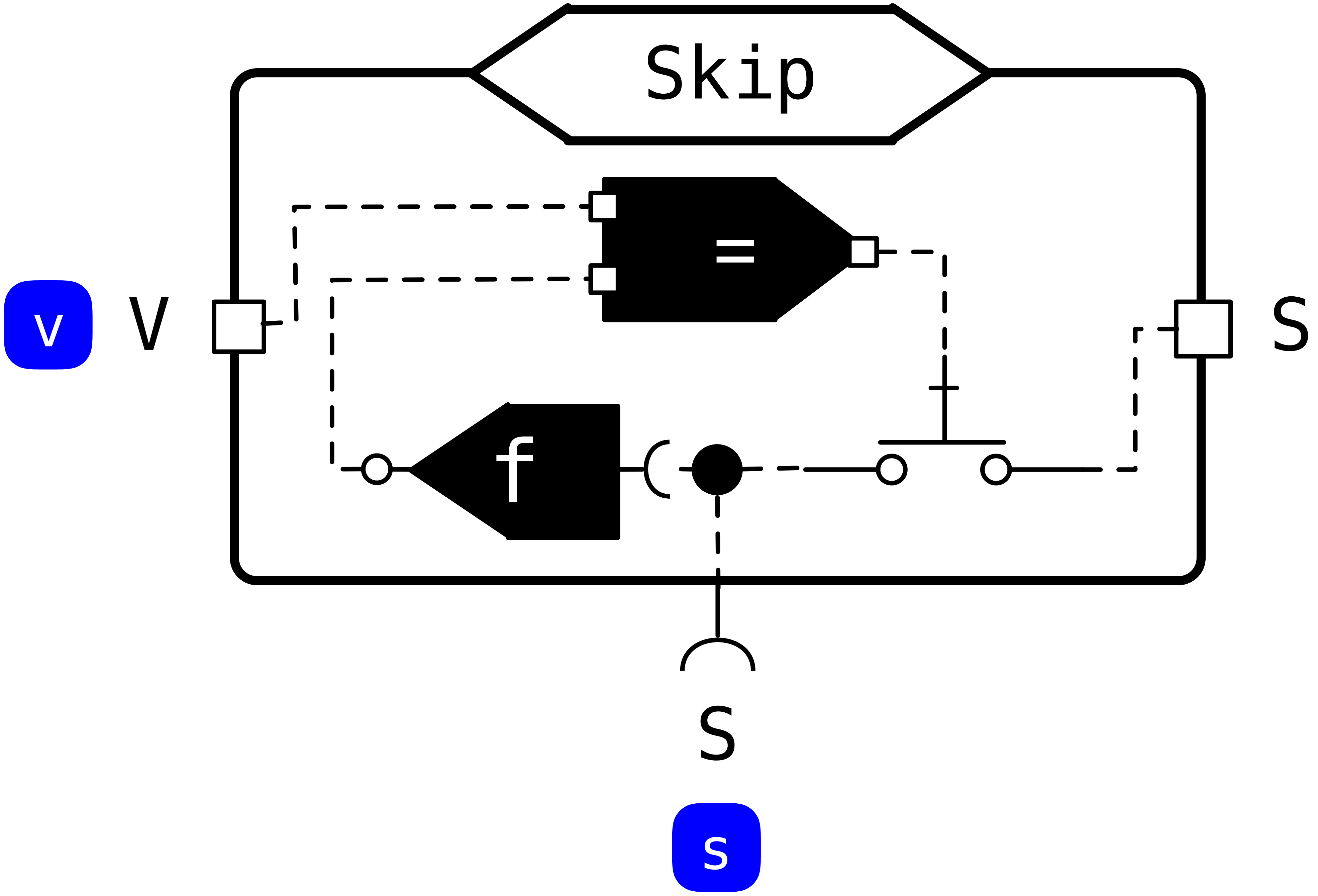
Replace



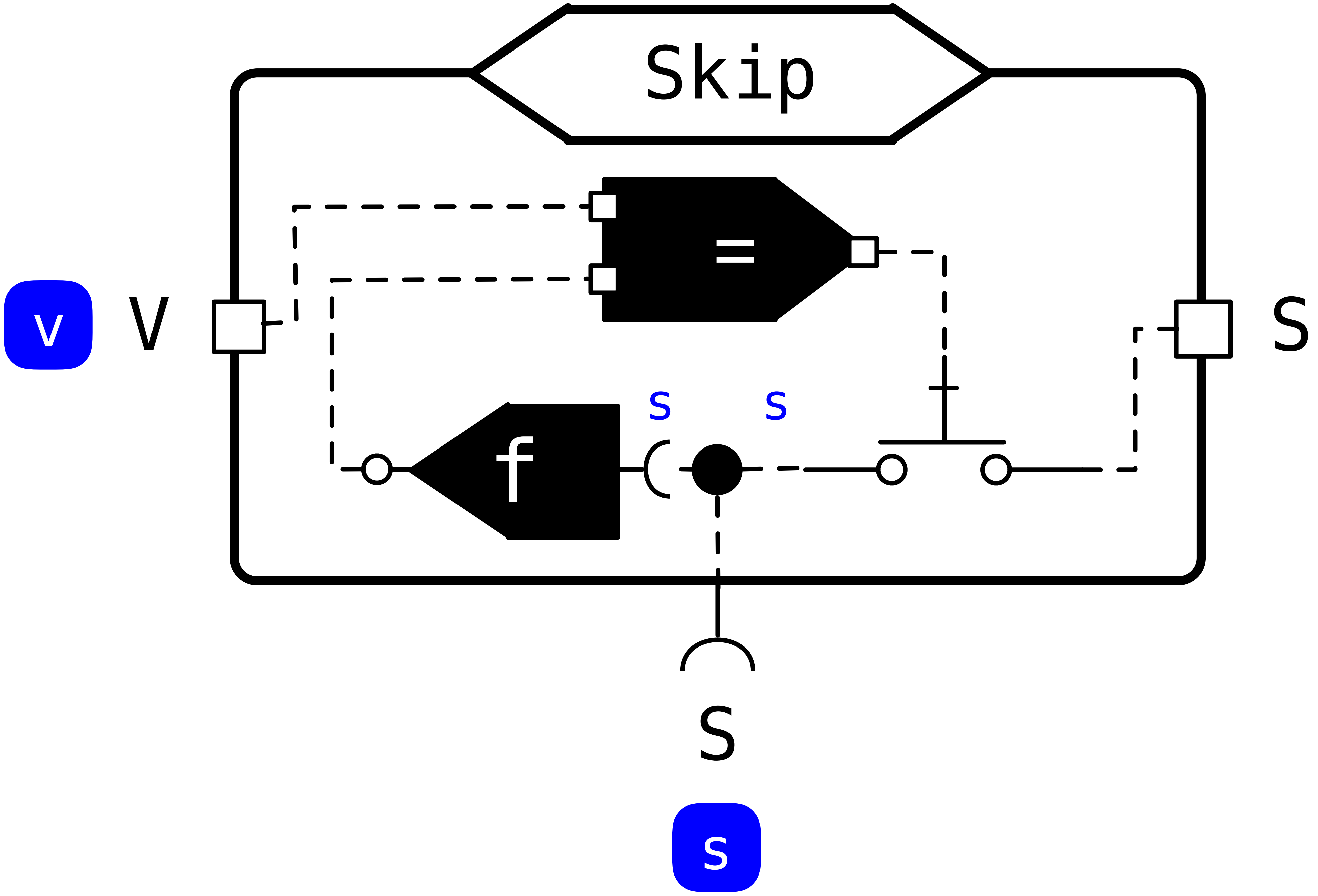
Skip



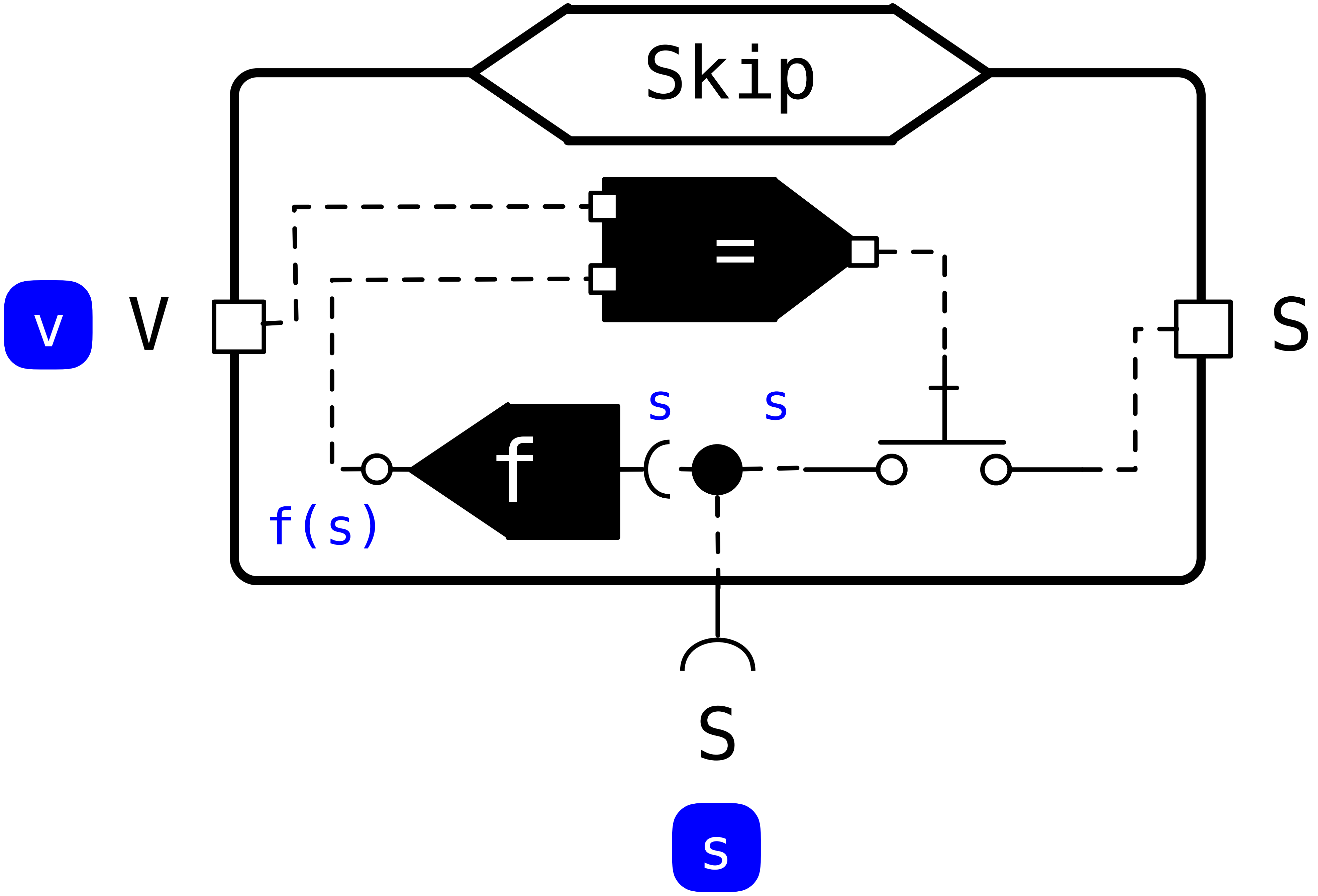
Skip



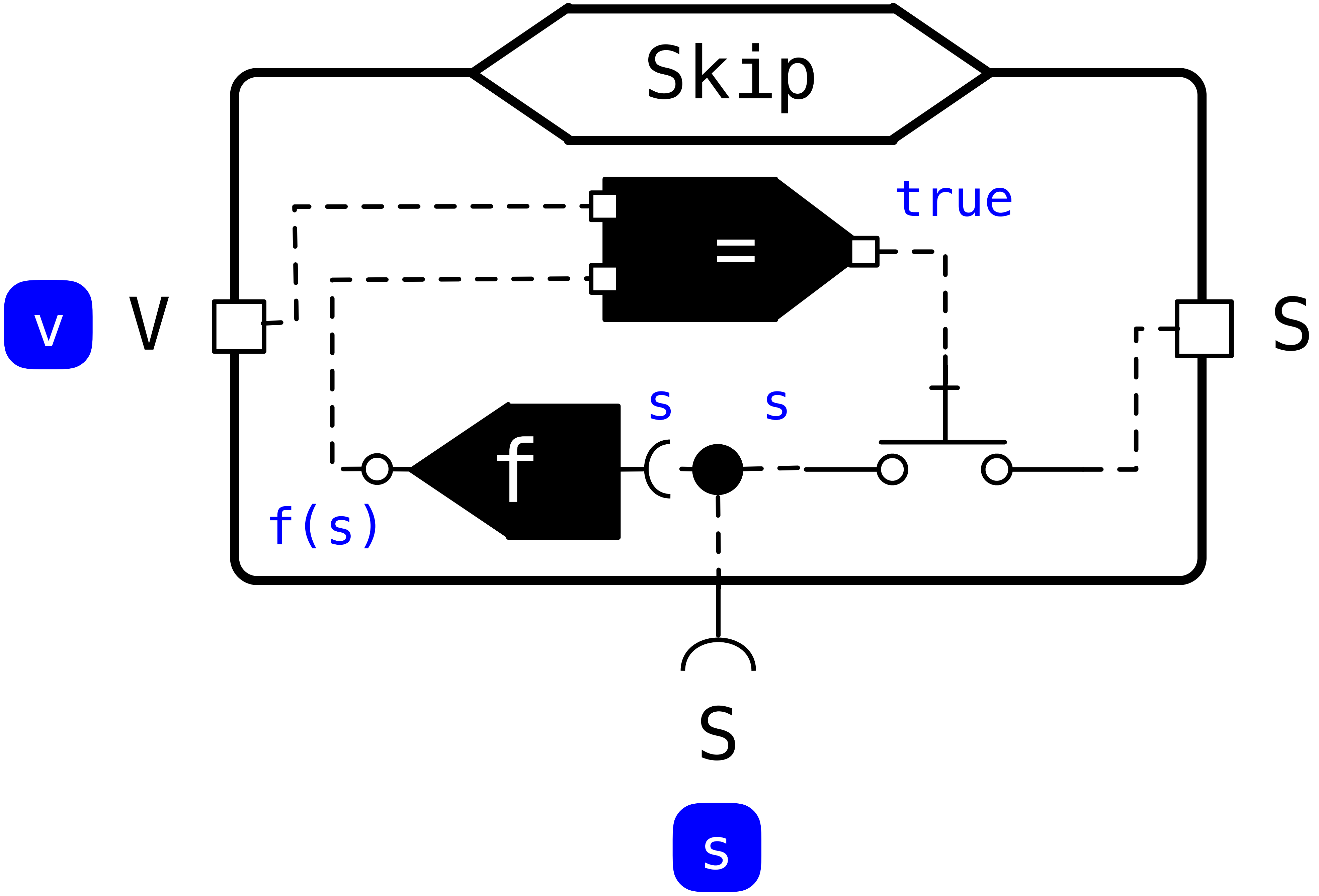
Skip



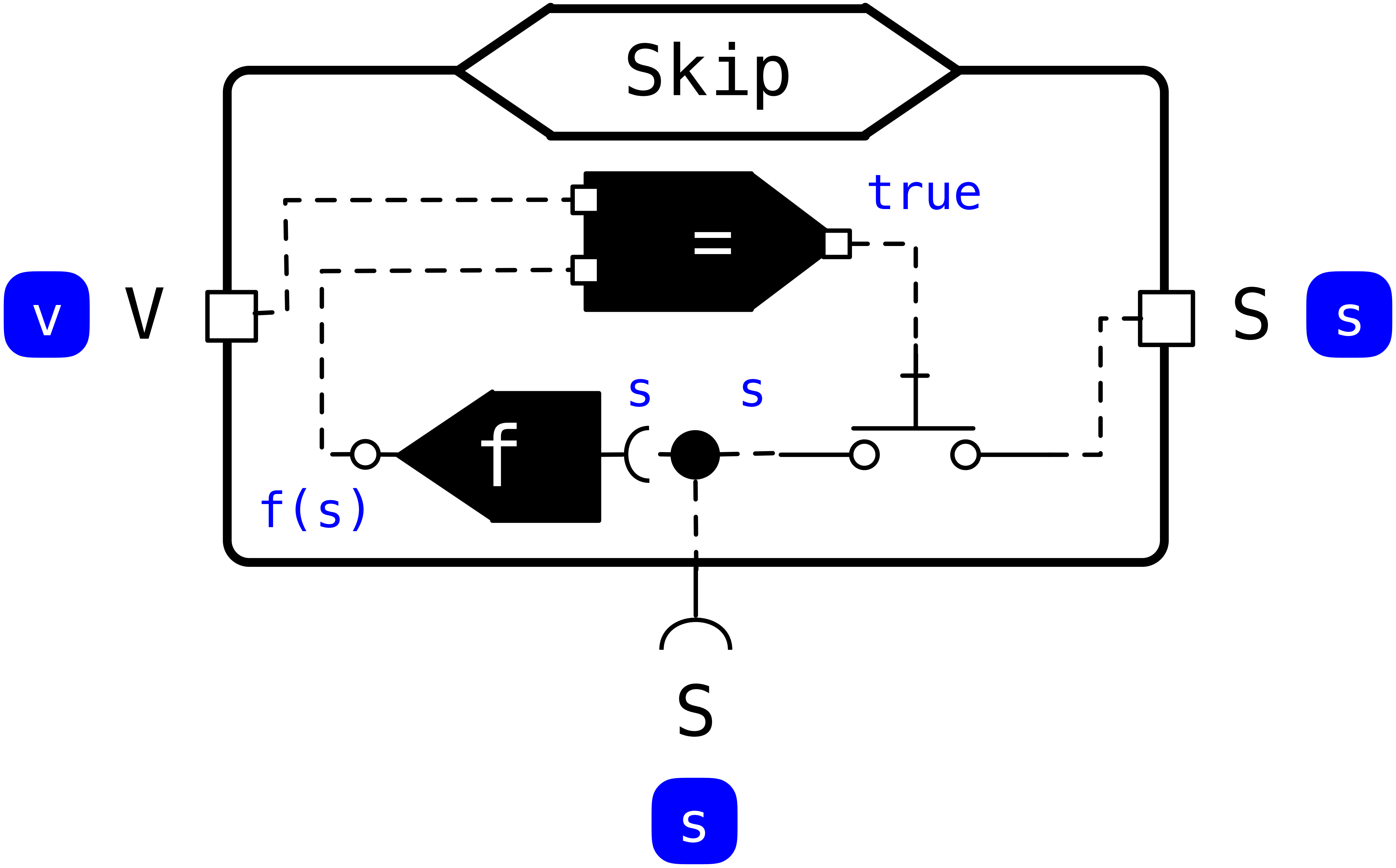
Skip



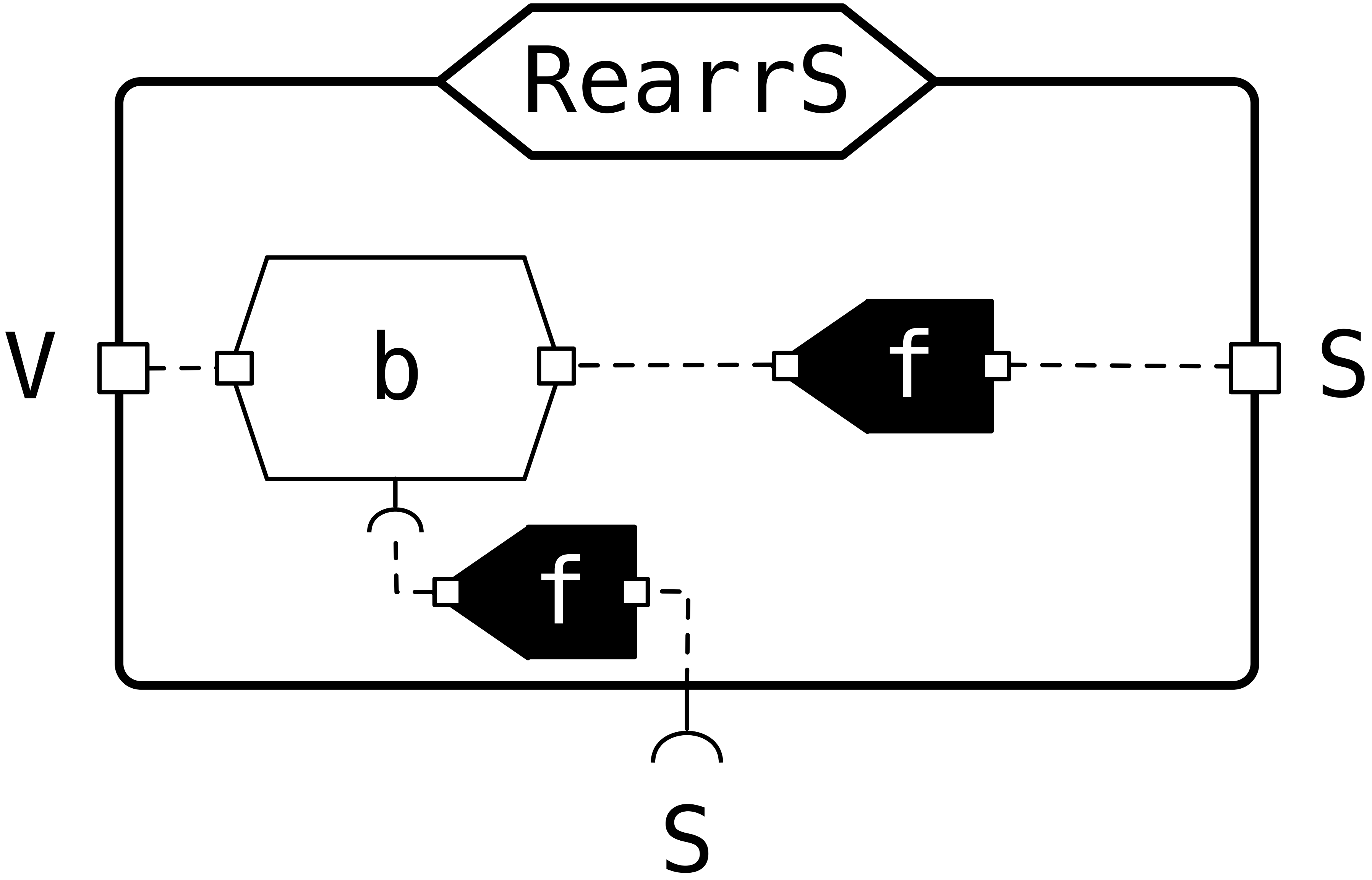
Skip



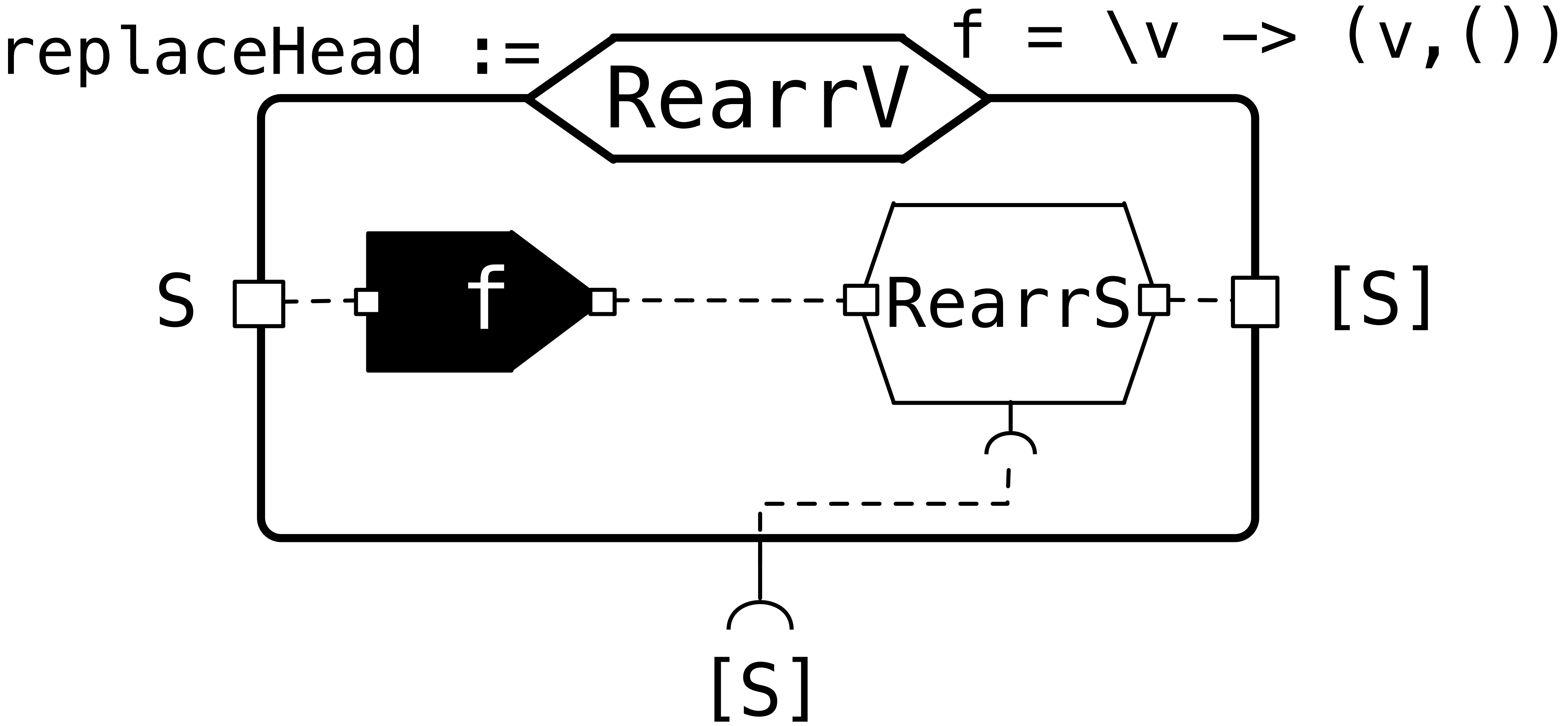
Skip



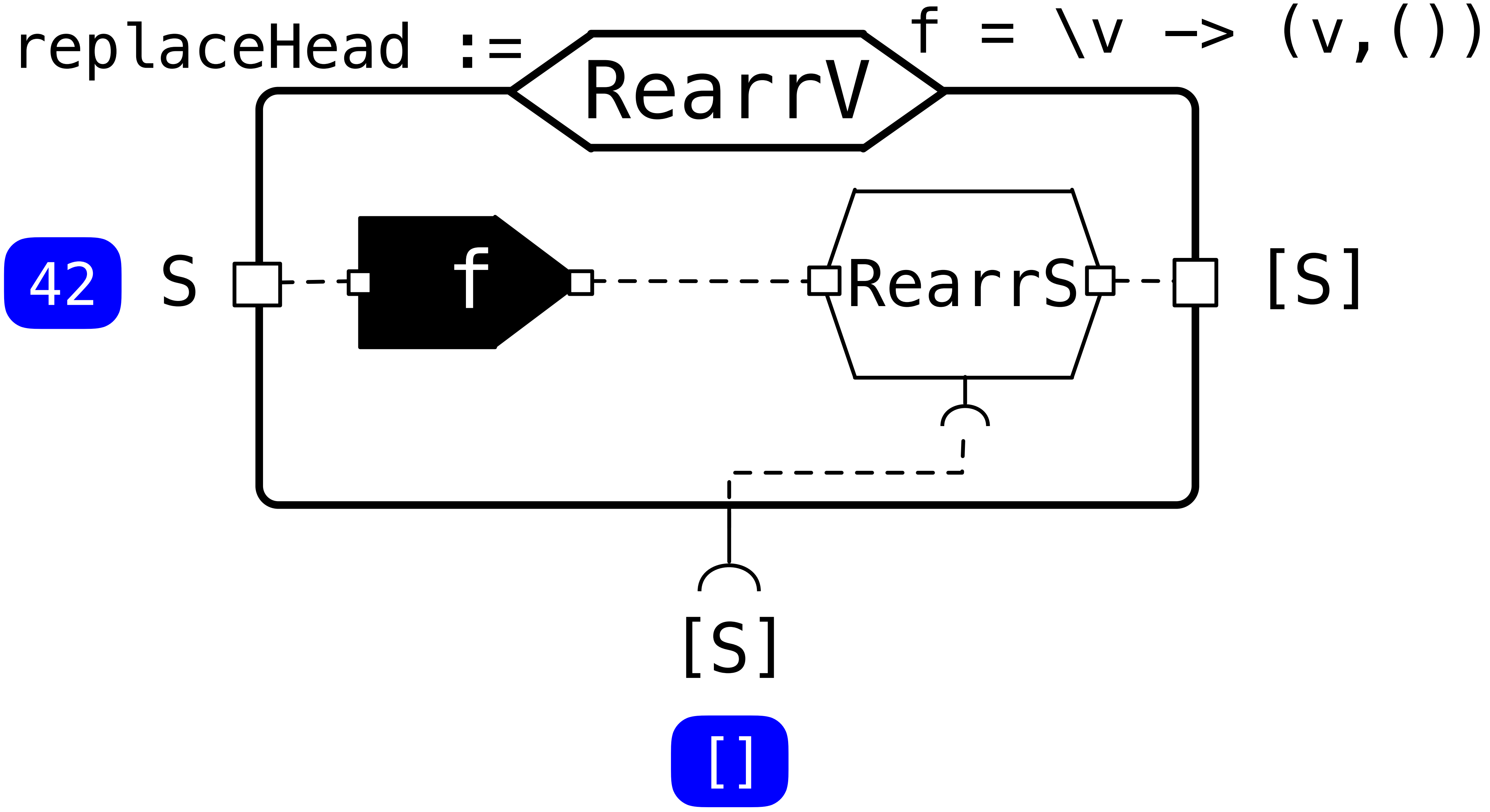
Source Rearrangement



Debugging



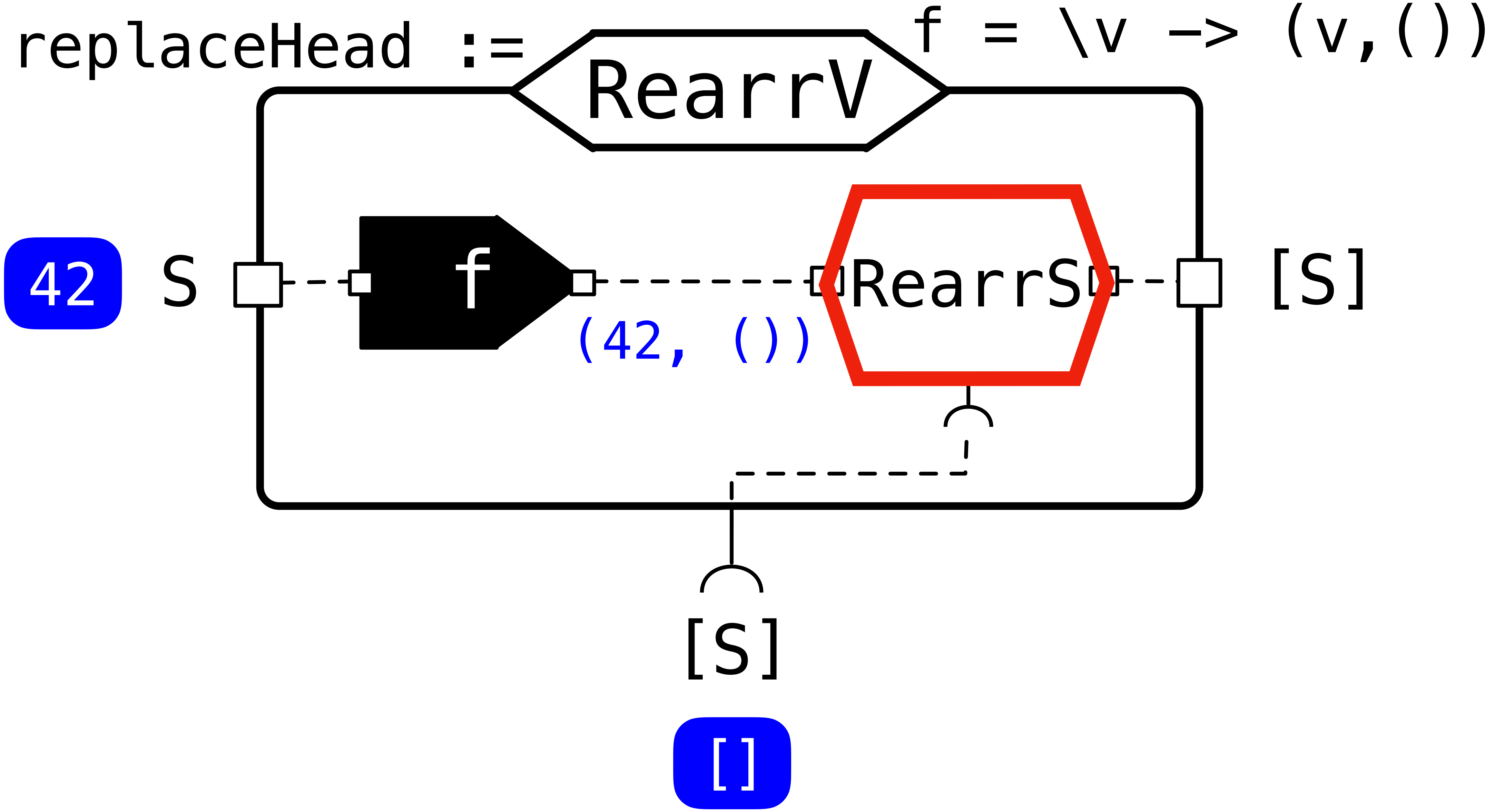
Debugging



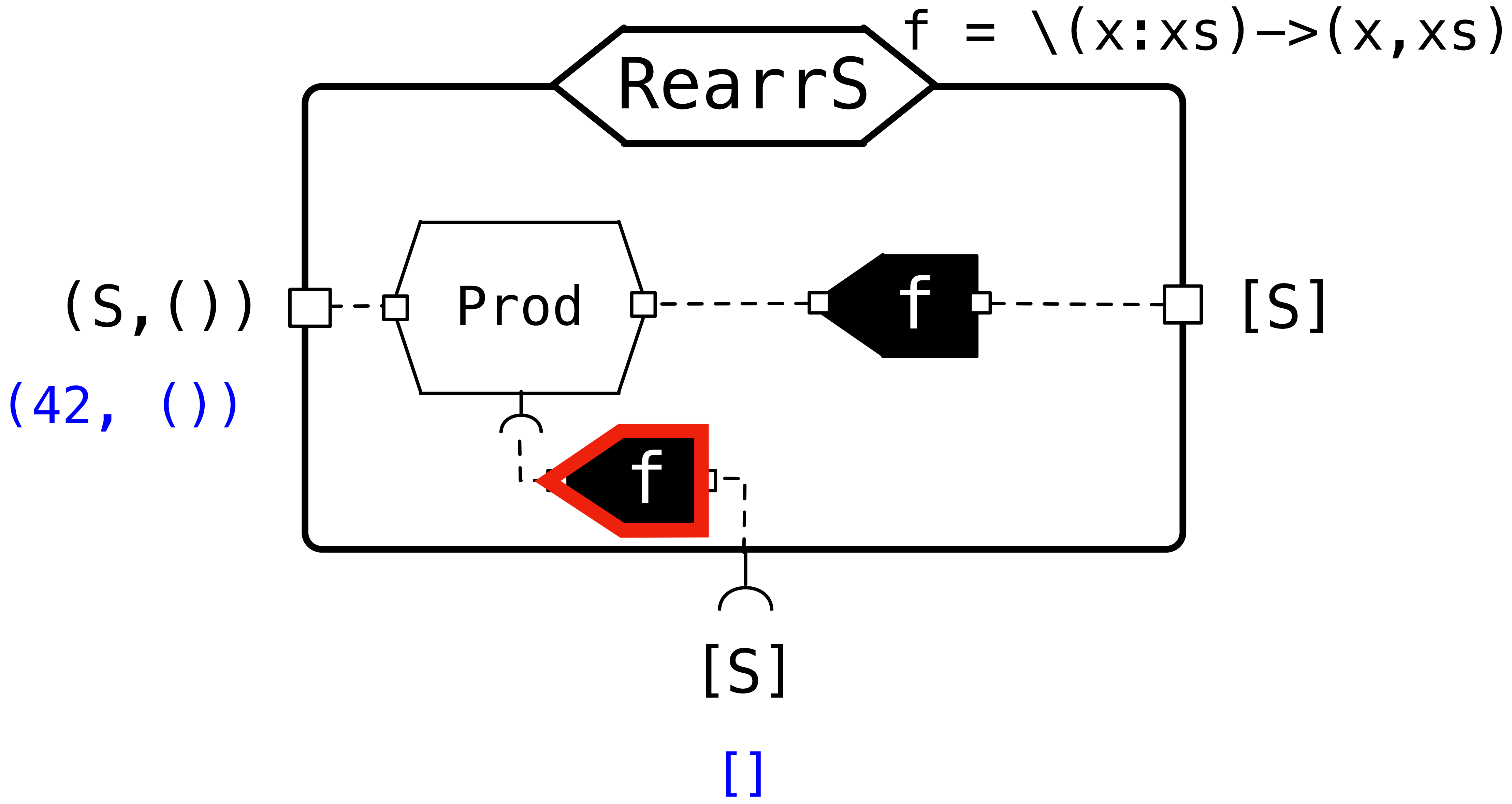
```
replaceHead :=
```

```
f = \v -> (v,())
```

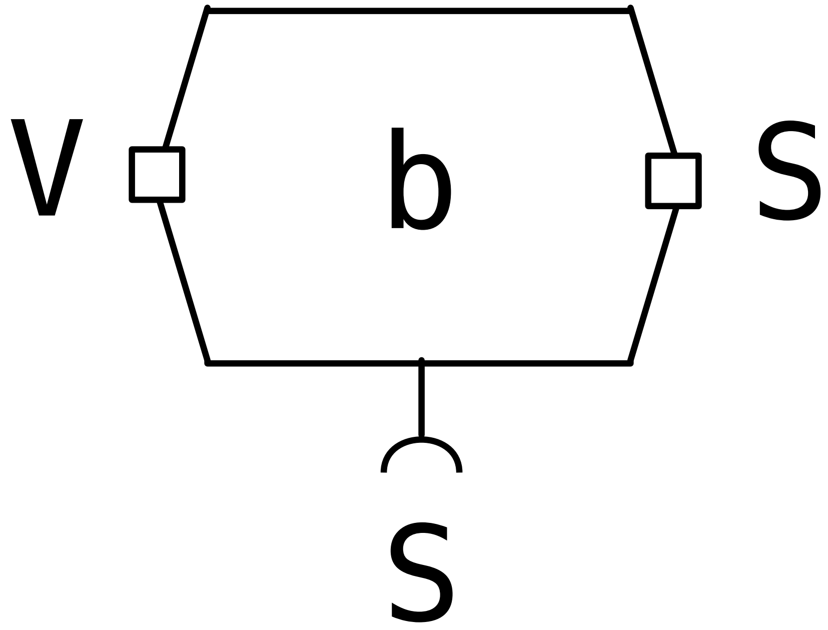
Debugging



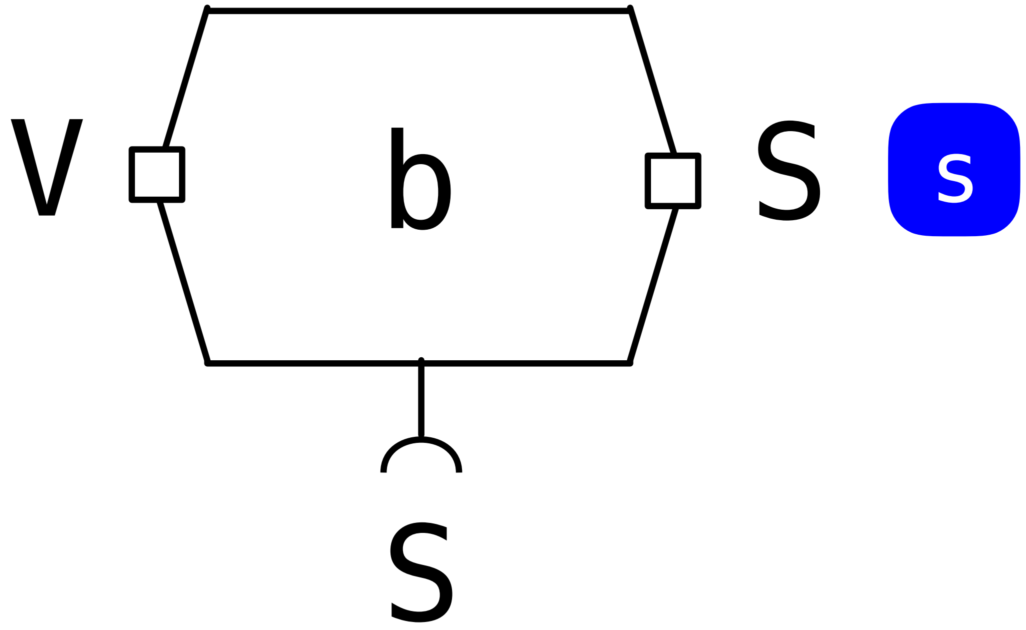
Debugging



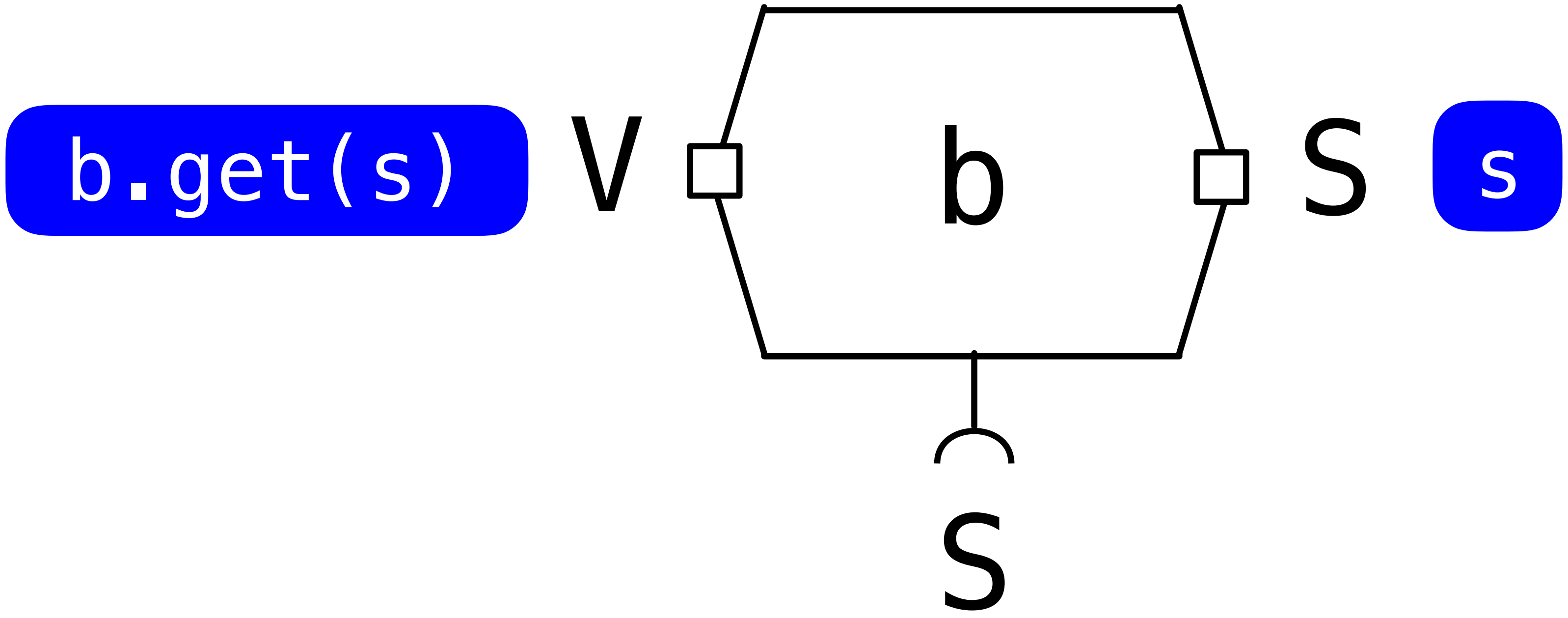
Get Semantics



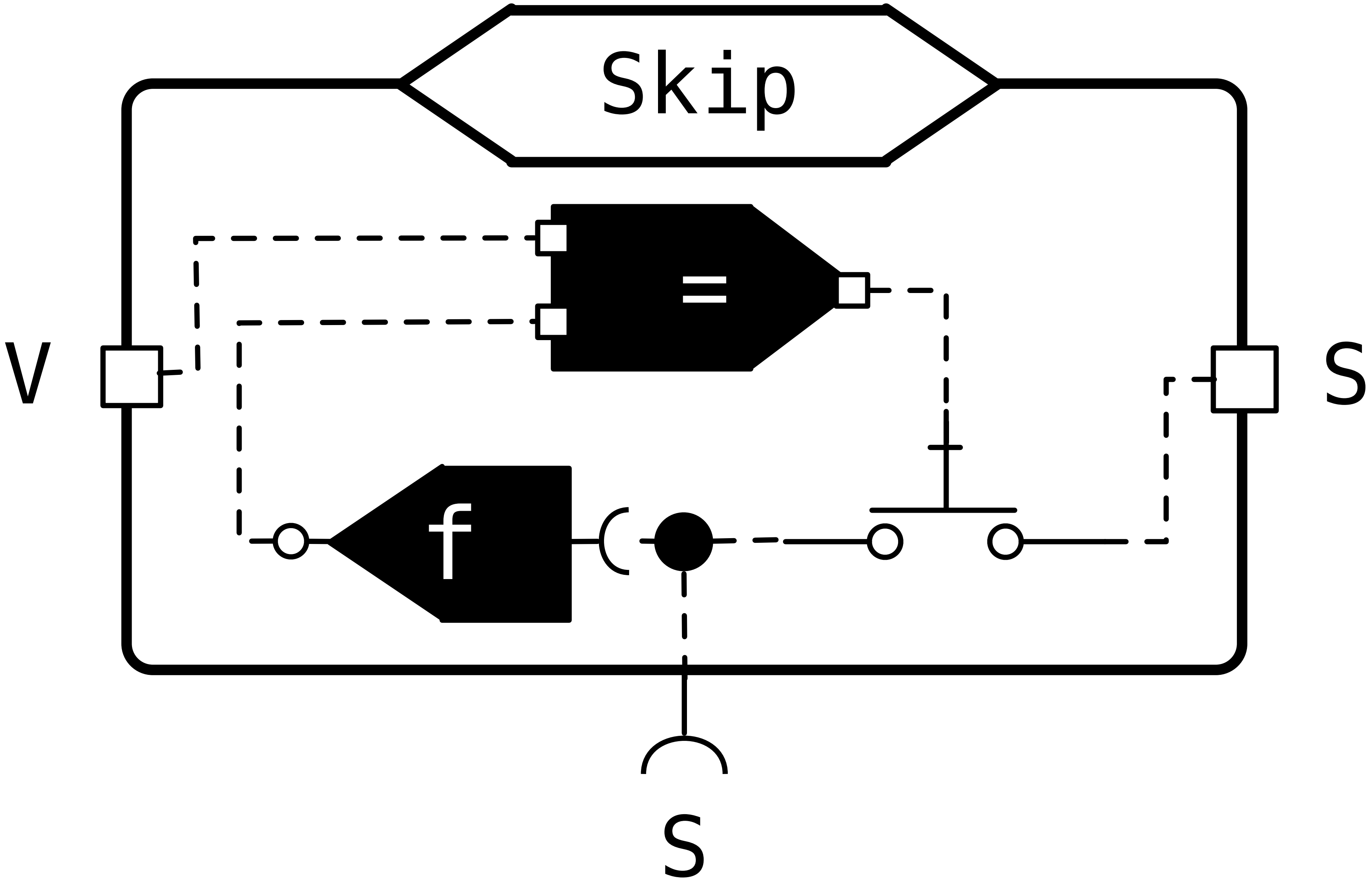
Get Semantics



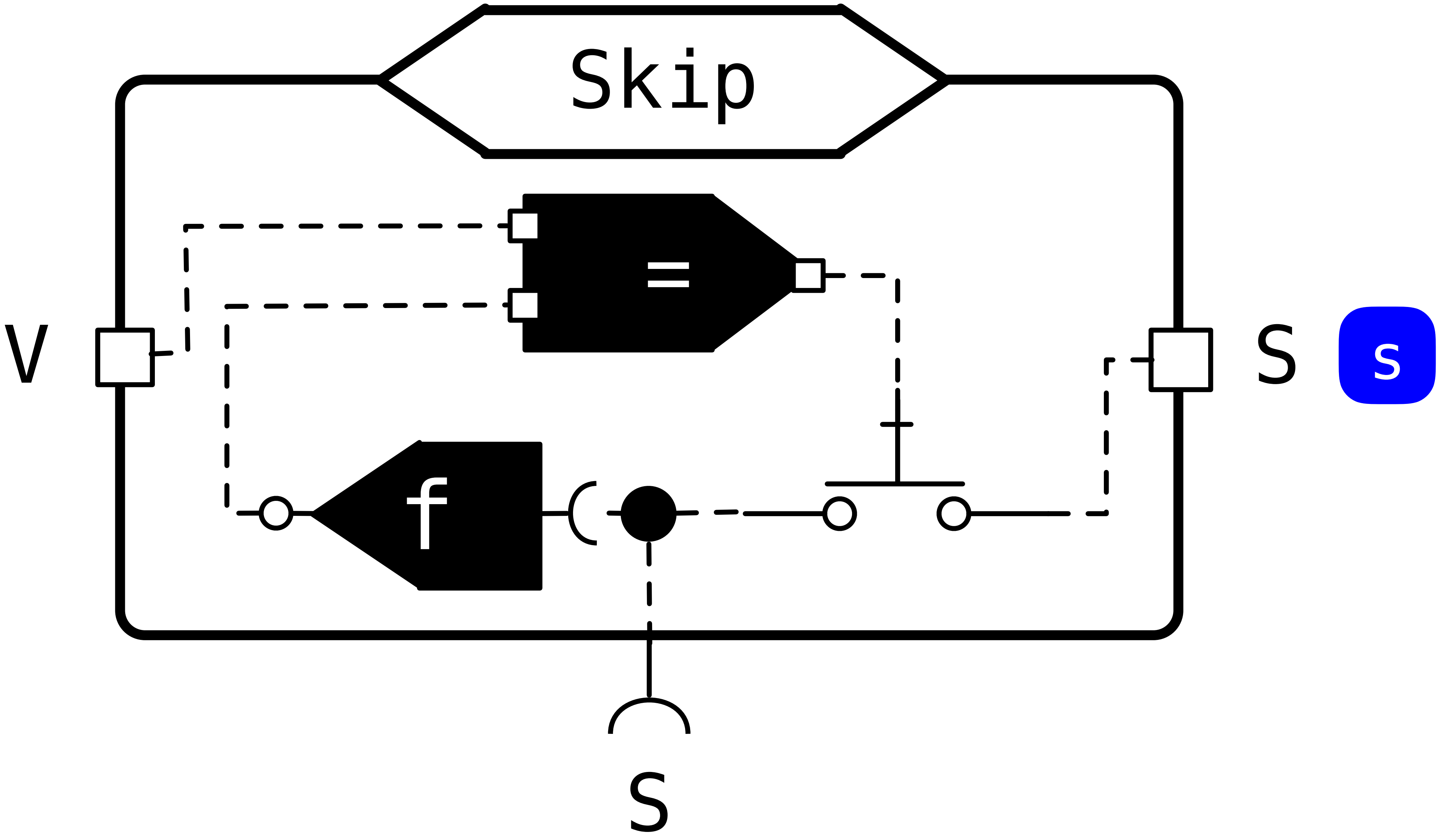
Get Semantics



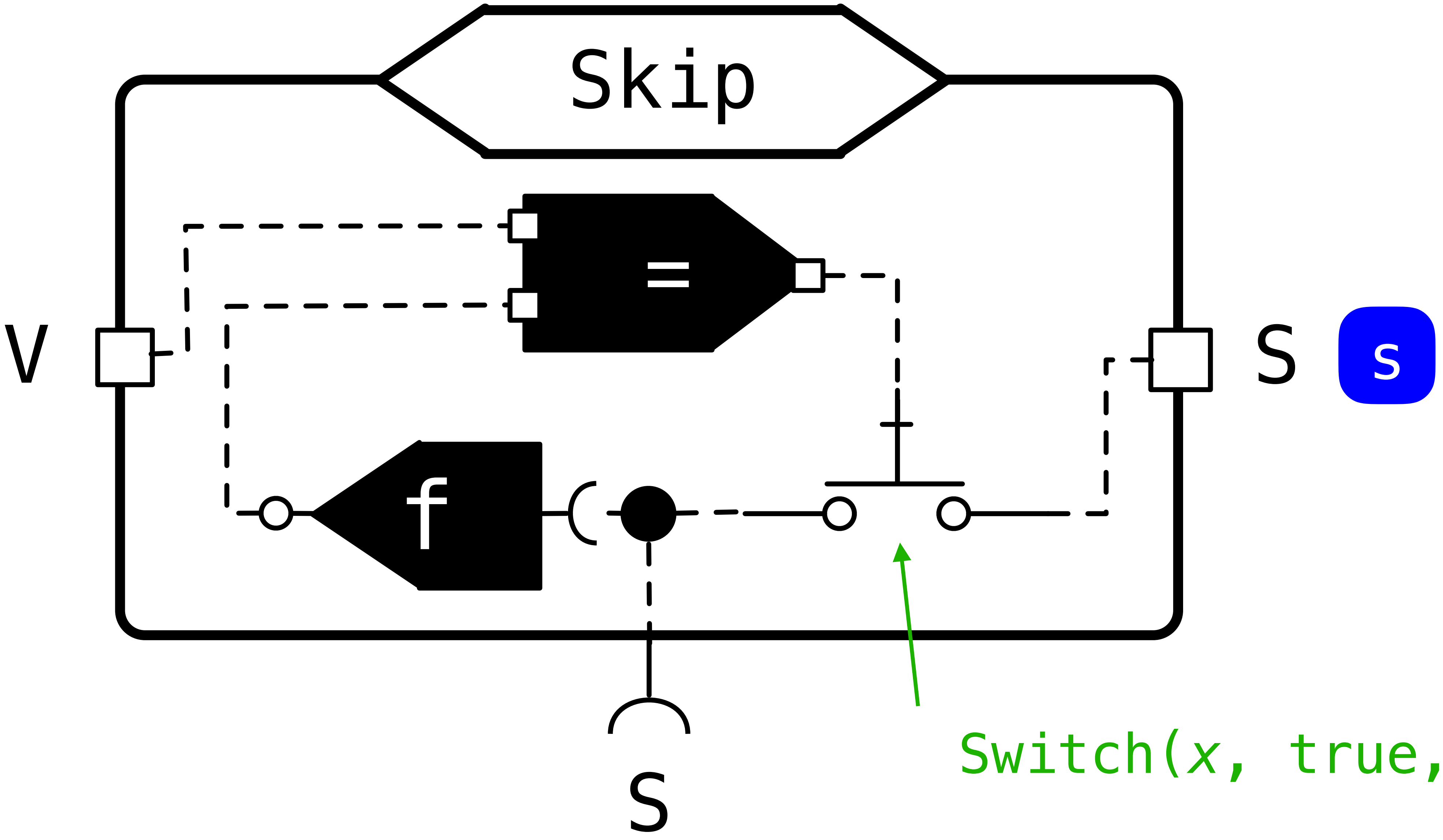
Skip (Get Semantics)



Skip (Get Semantics)

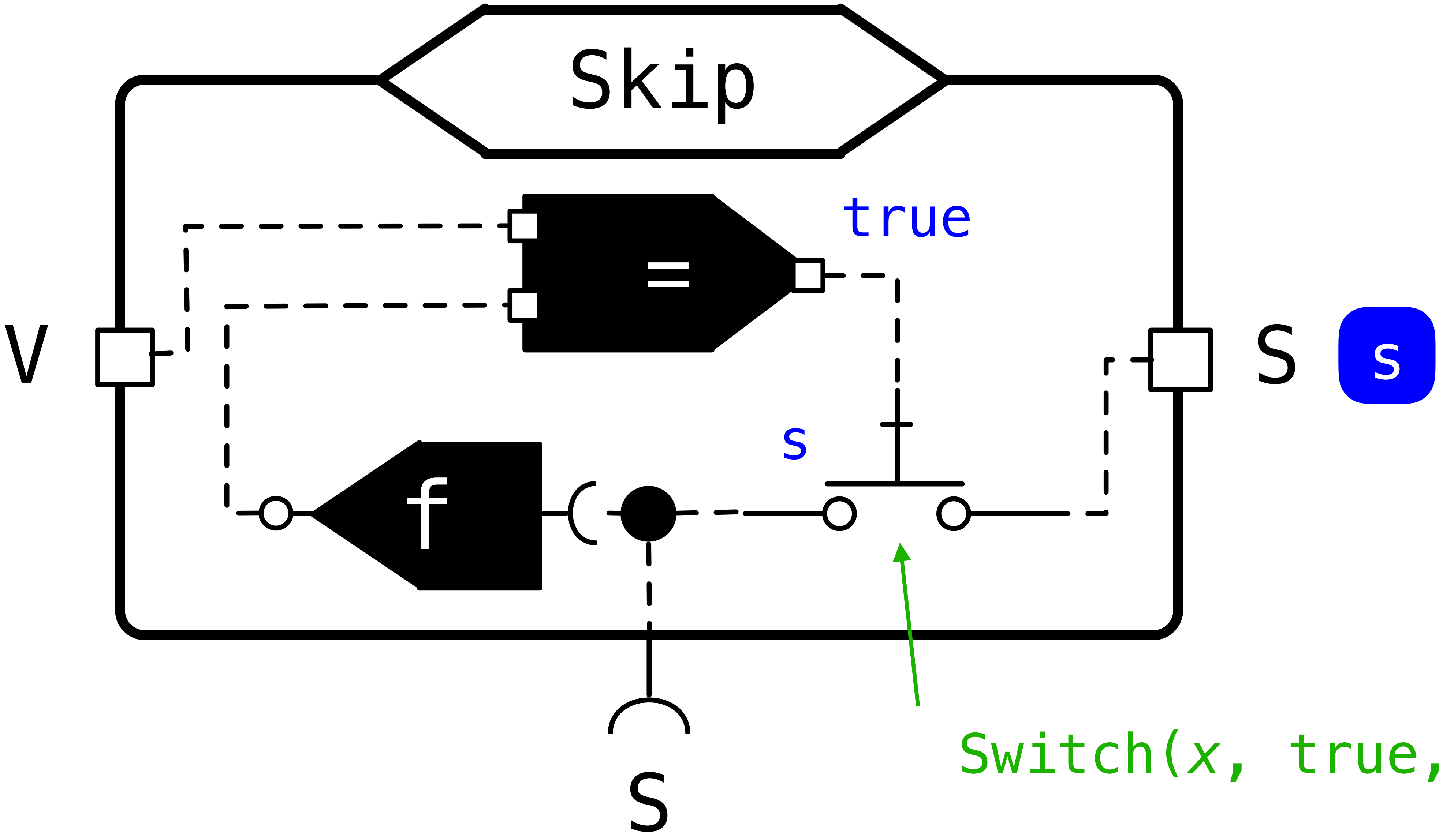


Skip (Get Semantics)



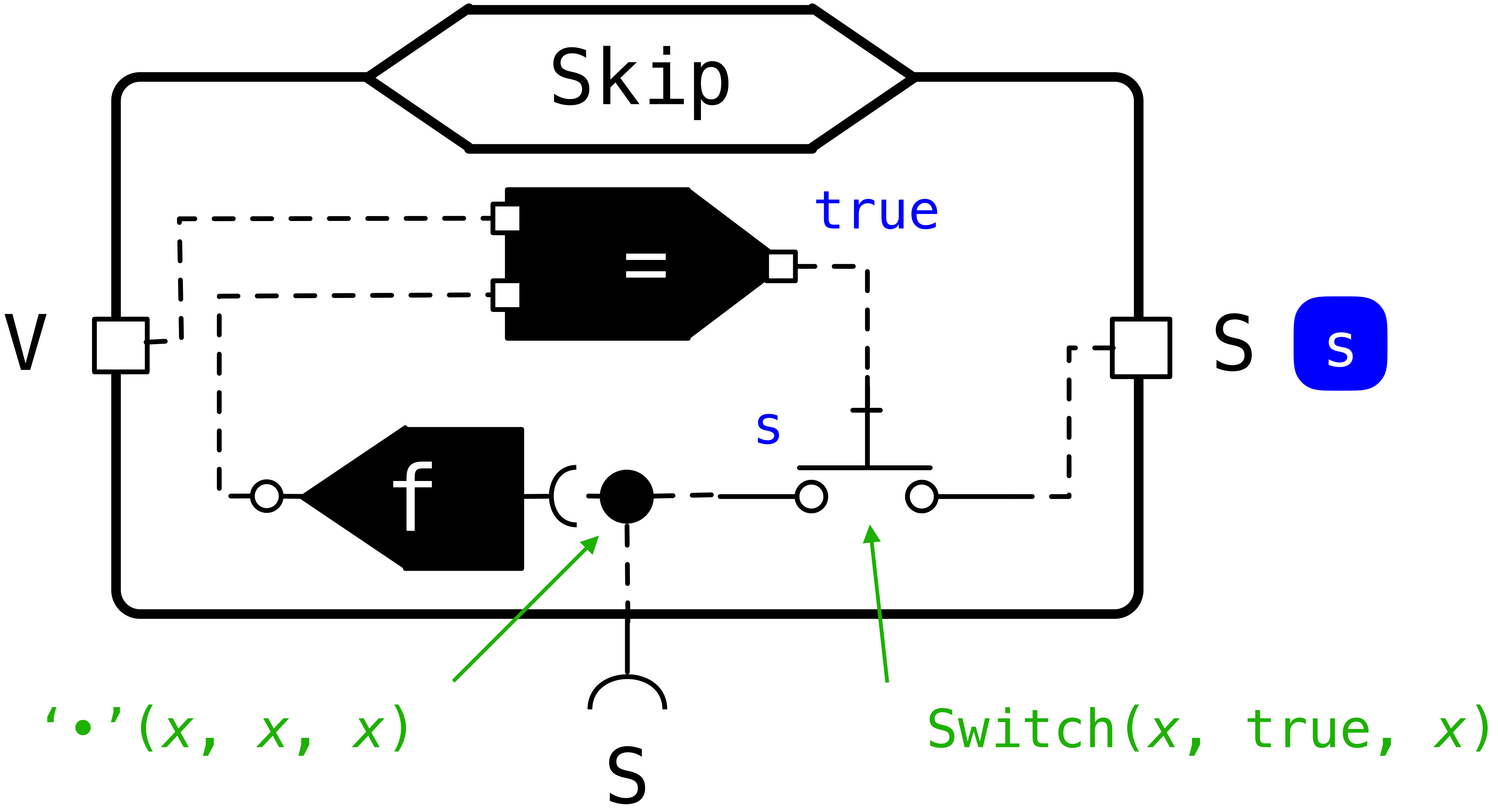
Switch(*x*, true, *x*)

Skip (Get Semantics)

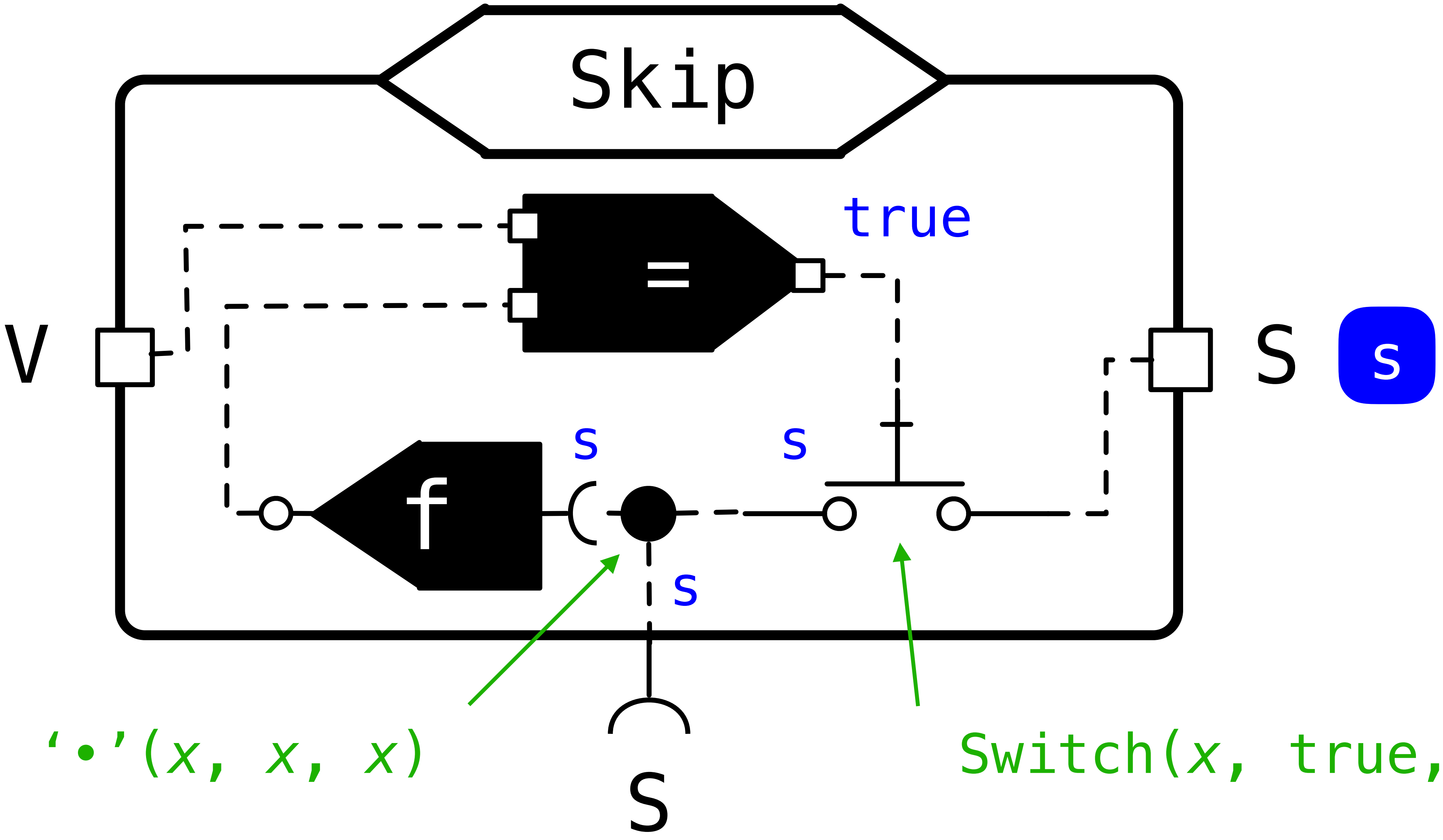


Switch(*x*, true, *x*)

Skip (Get Semantics)



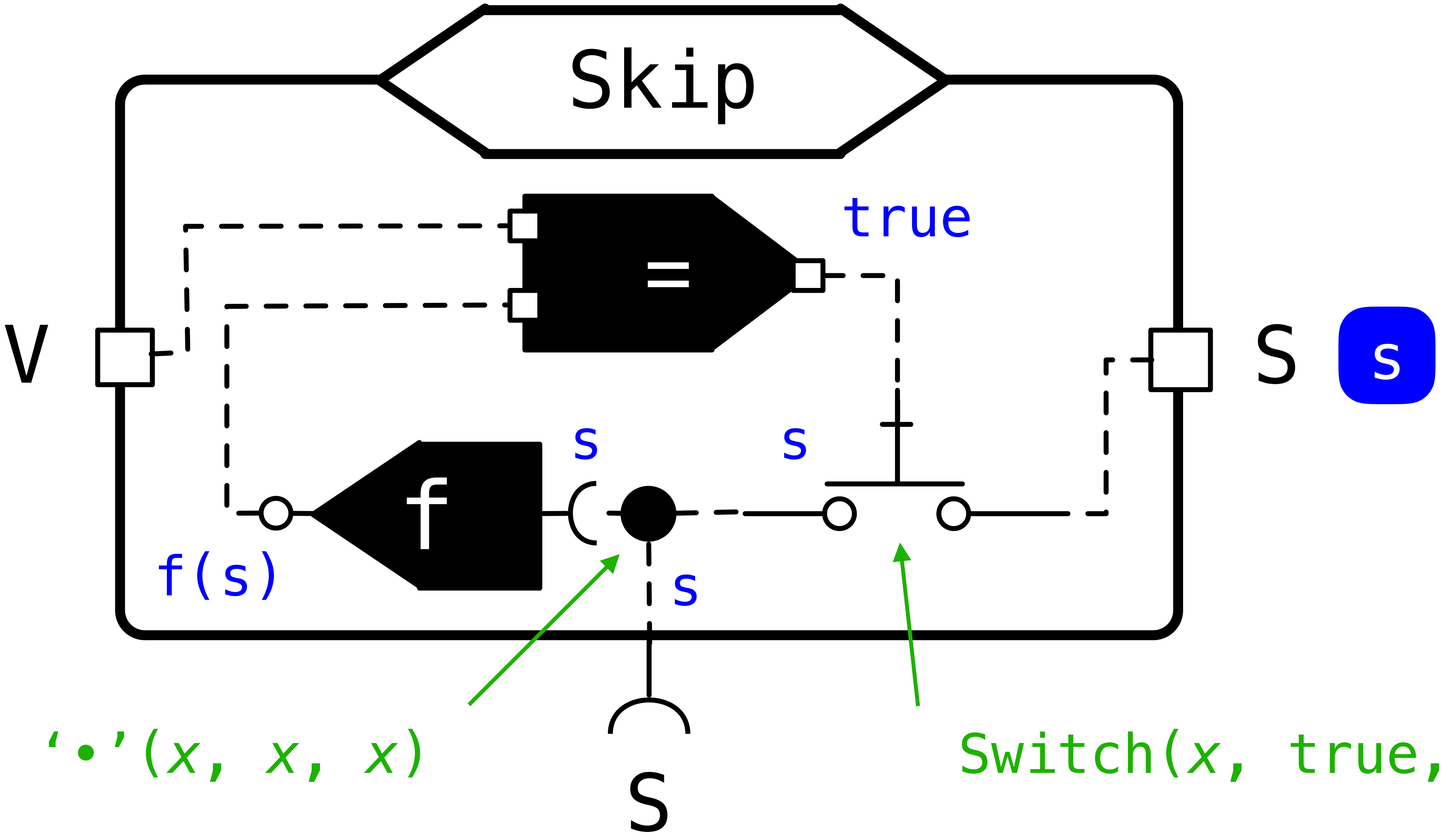
Skip (Get Semantics)



‘•’(*x*, *x*, *x*)

Switch(*x*, true, *x*)

Skip (Get Semantics)

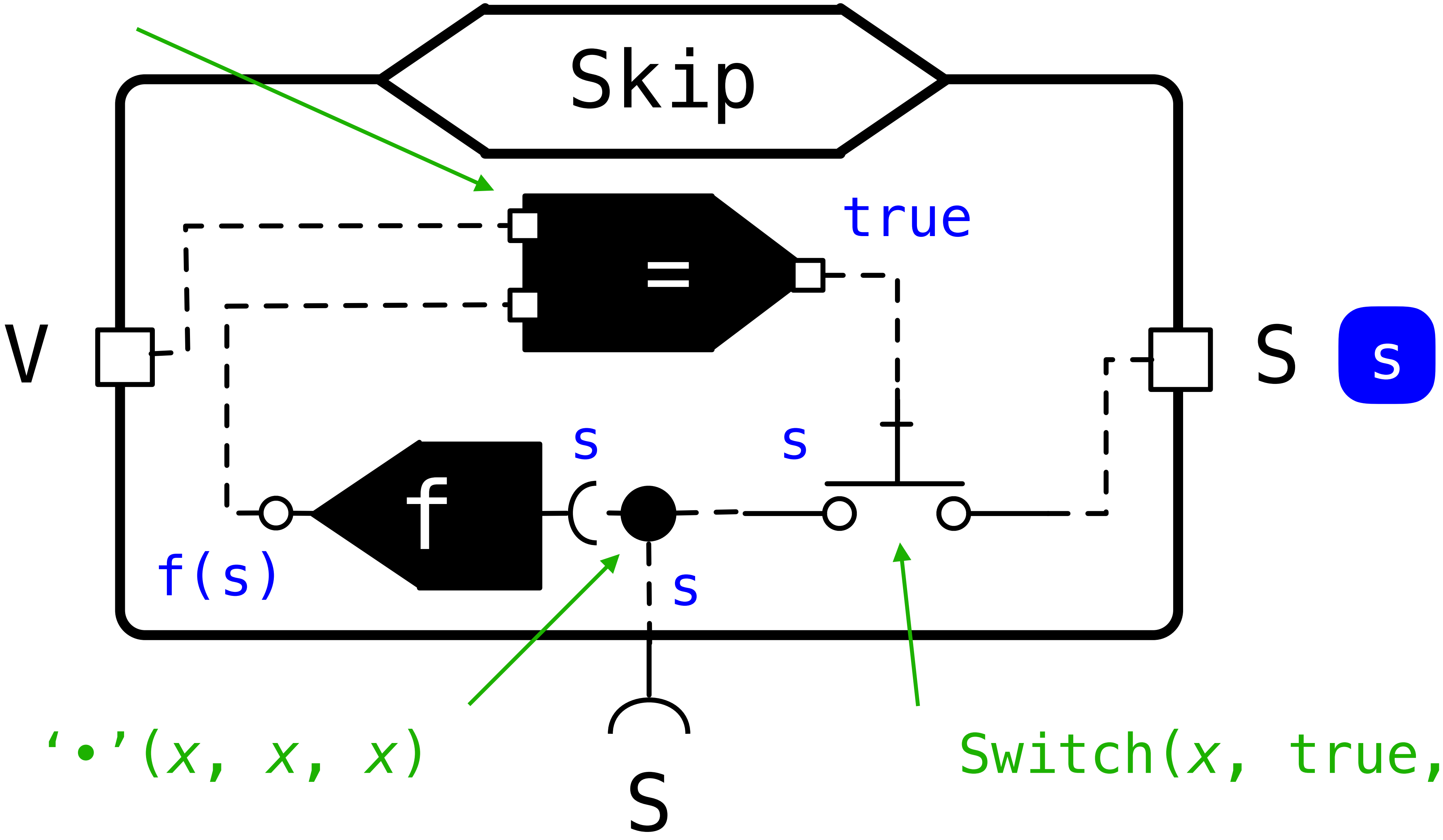


‘•’(*x*, *x*, *x*)

Switch(*x*, true, *x*)

Skip (Get Semantics)

‘=’(*x*, *x*, true)
‘=’(*x*, *y*, false) if *x* ≠ *y*

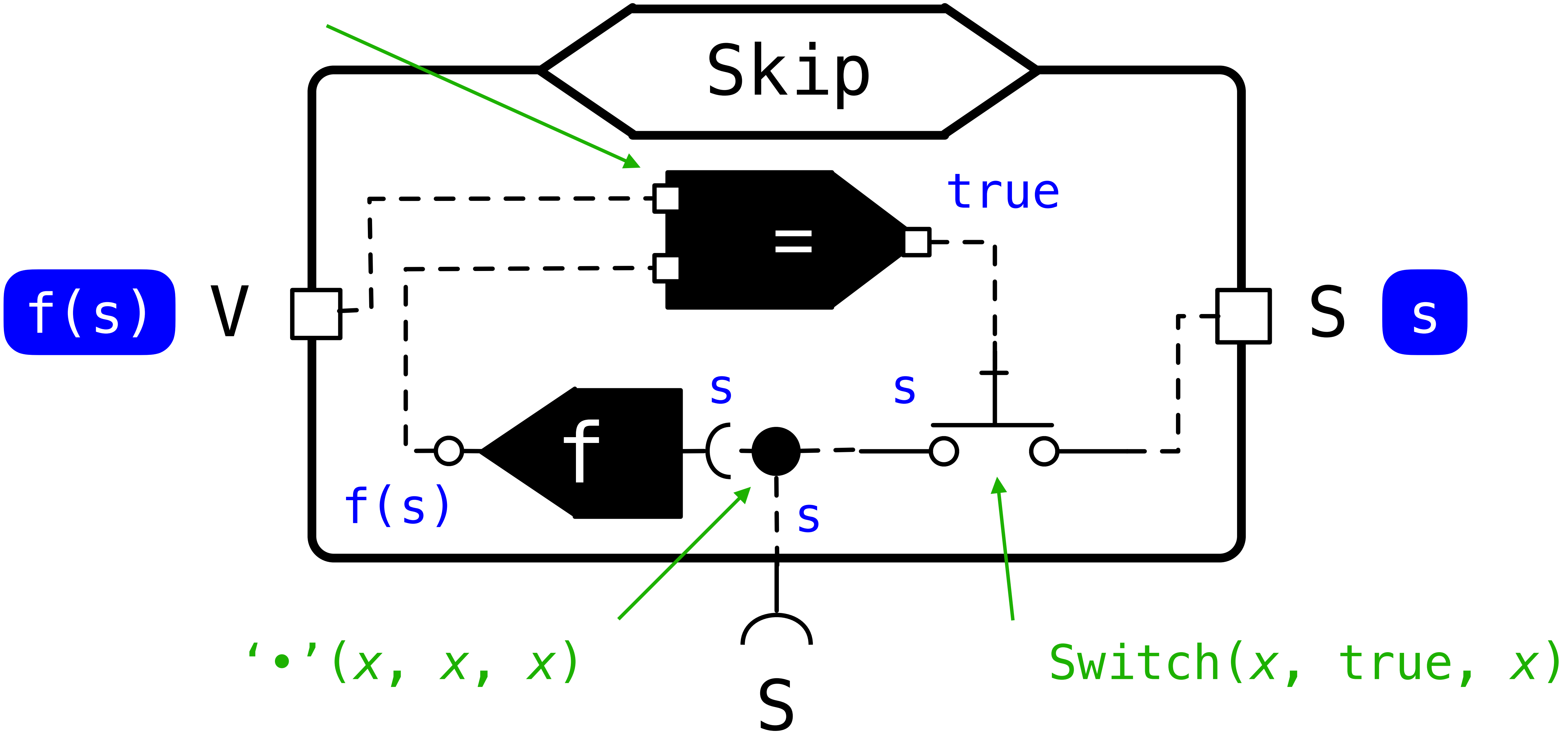


‘•’(*x*, *x*, *x*)

Switch(*x*, true, *x*)

Skip (Get Semantics)

‘=’(*x*, *x*, true)
‘=’(*x*, *y*, false) if *x* ≠ *y*



‘•’(*x*, *x*, *x*)

Switch(*x*, true, *x*)

How This Can Help

- A drag-and-drop visual editor, which is easier to use for programmers not familiar with Haskell
- Novice programmers often need to start from an operational understanding of the language.
- Proficient programmers sometimes also need to debug their program by tracing its execution.
  - BiGUL has an axiomatic semantics (to appear in the next session), which currently does not cover lens composition.

Beyond WB Combinators

- An instantiation of the relational/logic programming paradigm (?)
  - Lens combinators are deterministic in both directions.
- Well-behavedness has been regarded as an atomic property established and preserved by lens combinators, but the Skip circuit suggests that there is some “sub-atomic” structure to explore.
  - Prospect for “*deterministic relational programming*”?
    - Also subsuming reversible programming