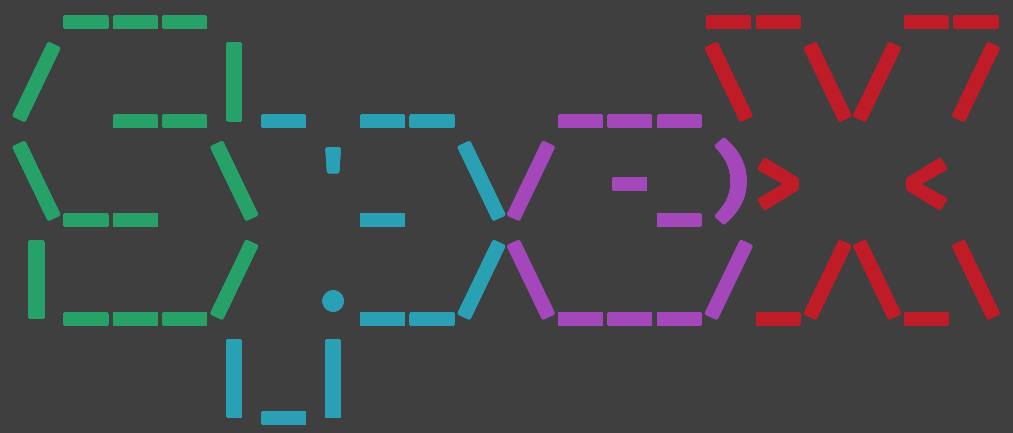


Ionuț Țuțu

IMAR, Romania

26th International Workshop on
Algebraic Development Techniques

In this talk

* interpreters (for free) via term rewriting
  ↳ of specification languages

* based on formal-specification technologies

  - sorts `sort Nat .`
  - subsorts `subsort Nat < Int .`
  - operations `op _+_ : Nat Nat → Nat .`
  - equations `eq X + 0 = X .`
  - rules `crl X, Y ⇒ Y, X if Y < X .`

Term rewriting as a substructure for specification-language interpreters

* solid mathematical foundation
* algebraic, close to standard notation used by working theoretical computer scientists
* great for rapid prototyping

but

* still somewhat rigid as a meta-language
* limited support for modularization

Object-based programming

* we rewrite configurations
  multisets of ↲

  1. objects `< Id : Class | Attributes >`
  2. messages `message(To, From, Arguments)`

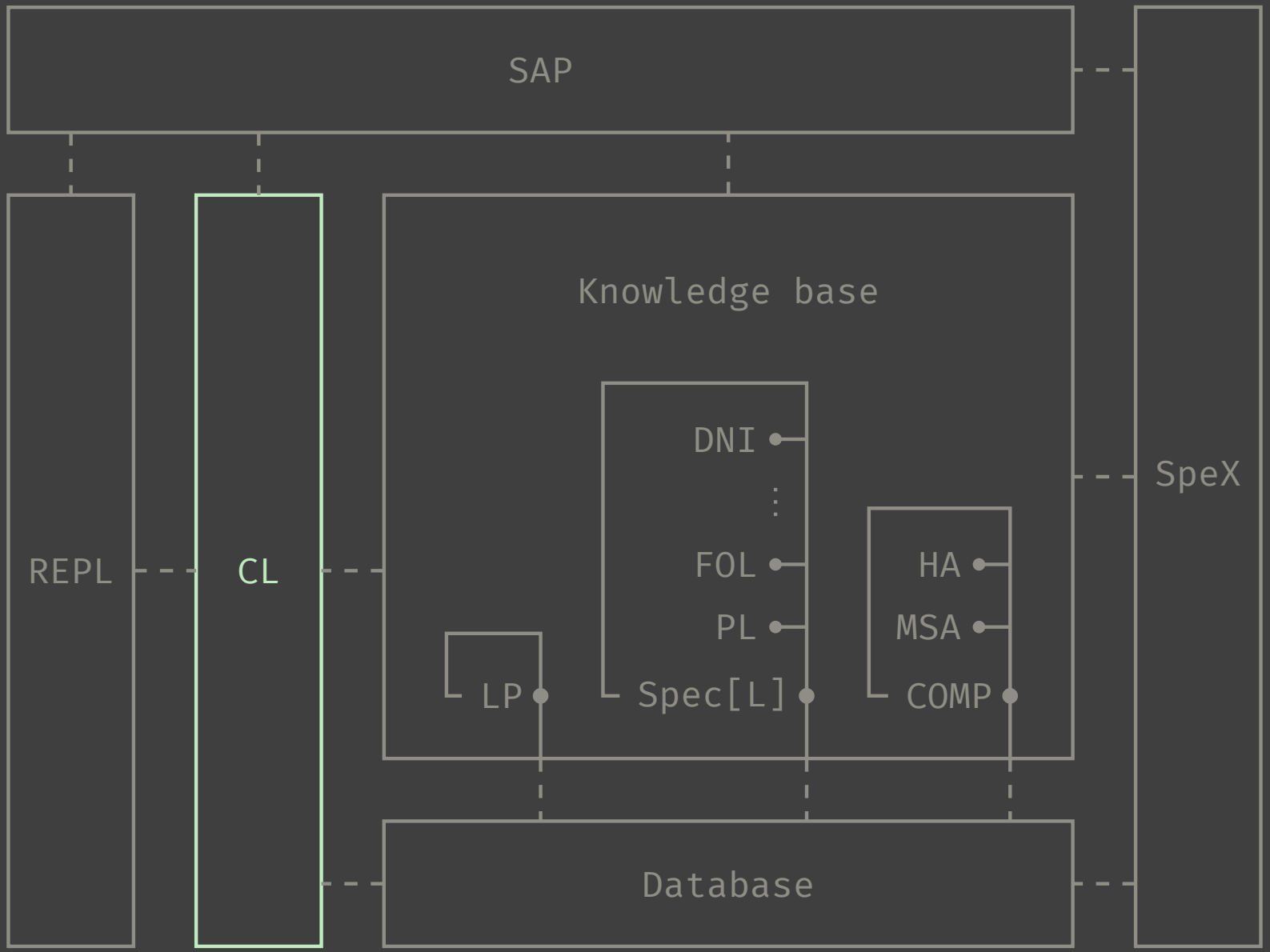
* using rules of the form

```
rl < Id : Class | ... >
   message(Id, ...)
  ⇒ ...
```

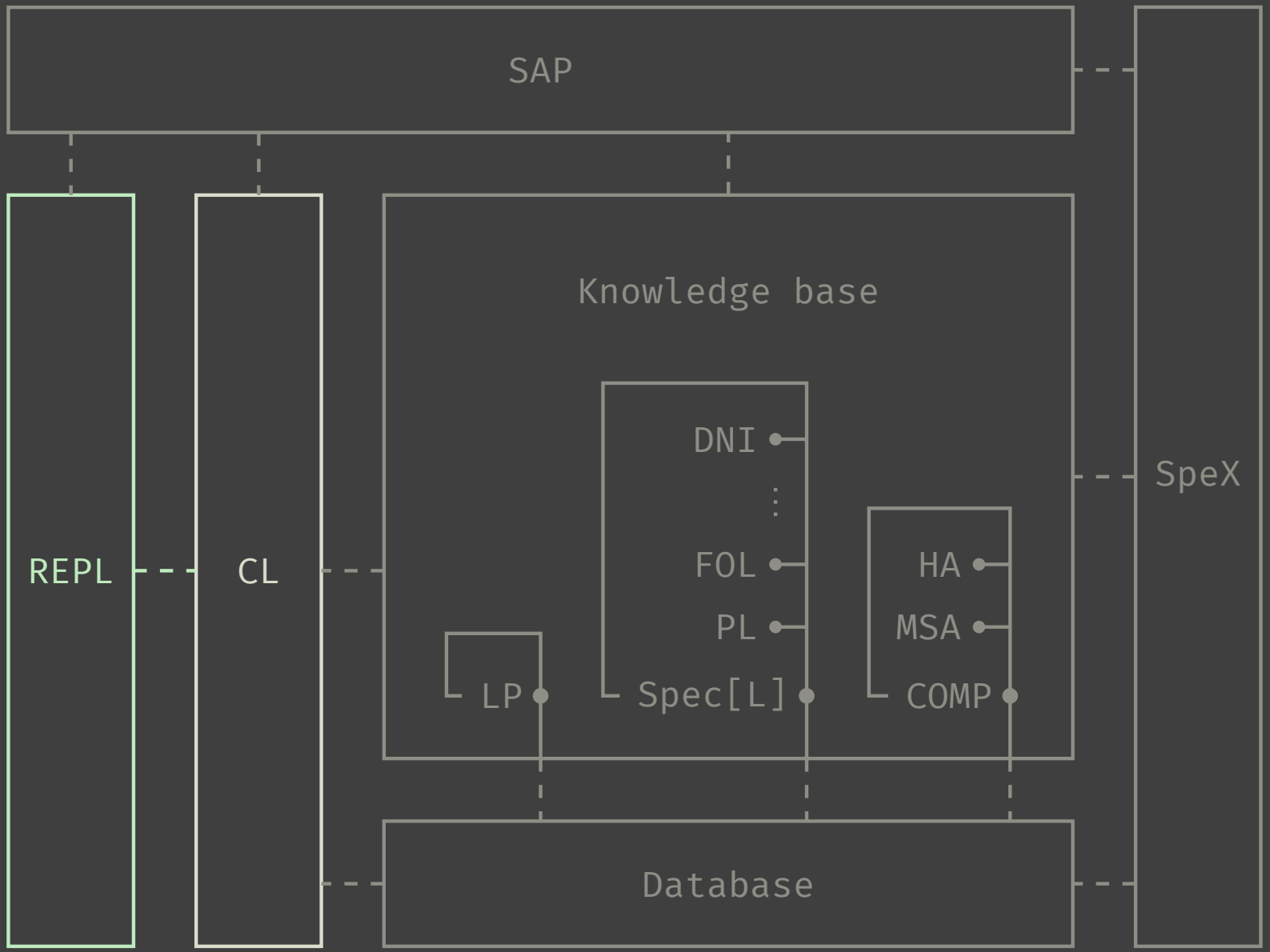
Introducing SpeX

* not a plain interpreter, but an ‘environment’
* integrates specification-language processors
* language agnostic
* offers a basic system UI ‘for free’
* based on Maude 3 (OBP with external rewrites)

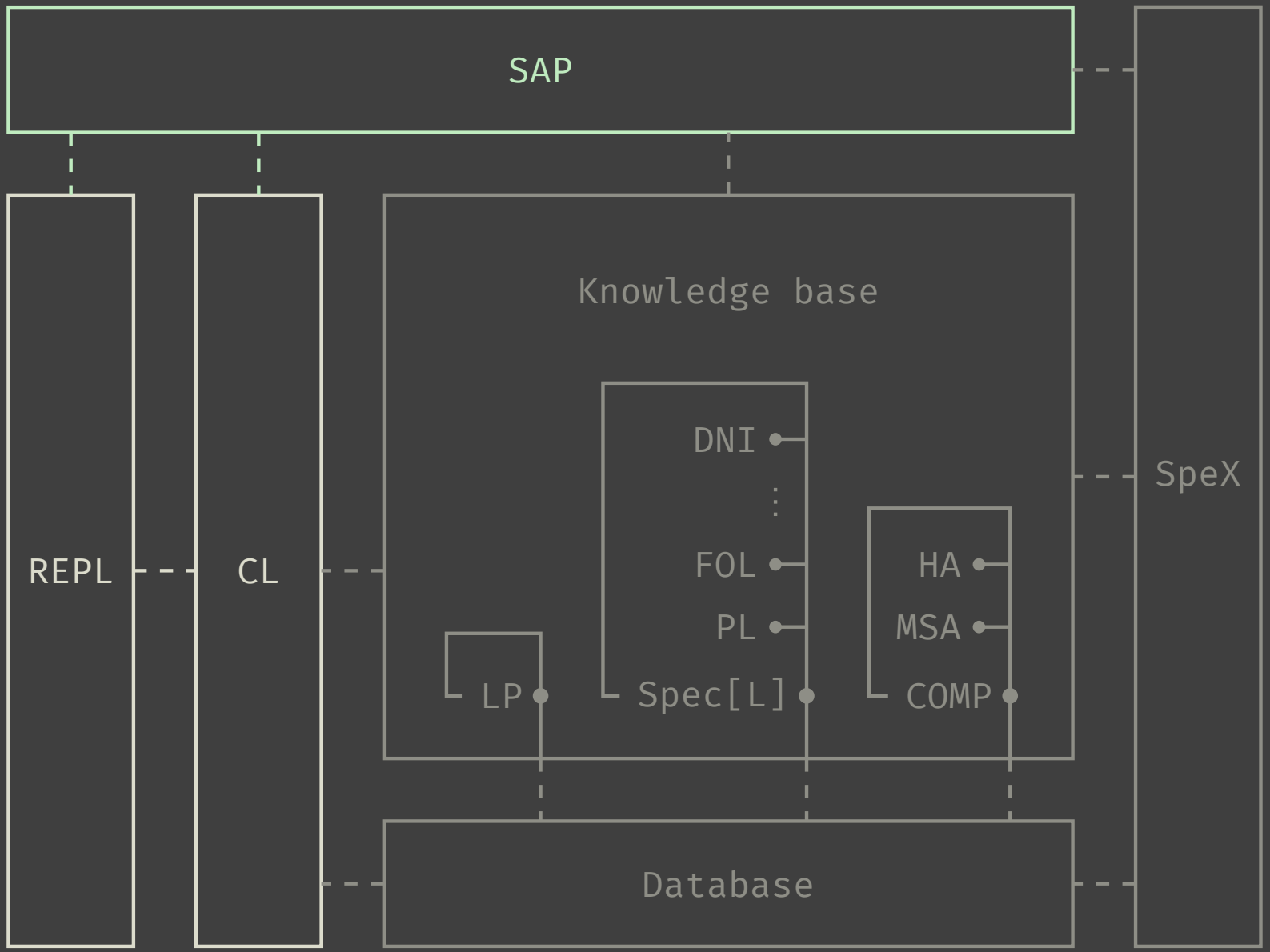
An overview of SpeX (overly simplified)



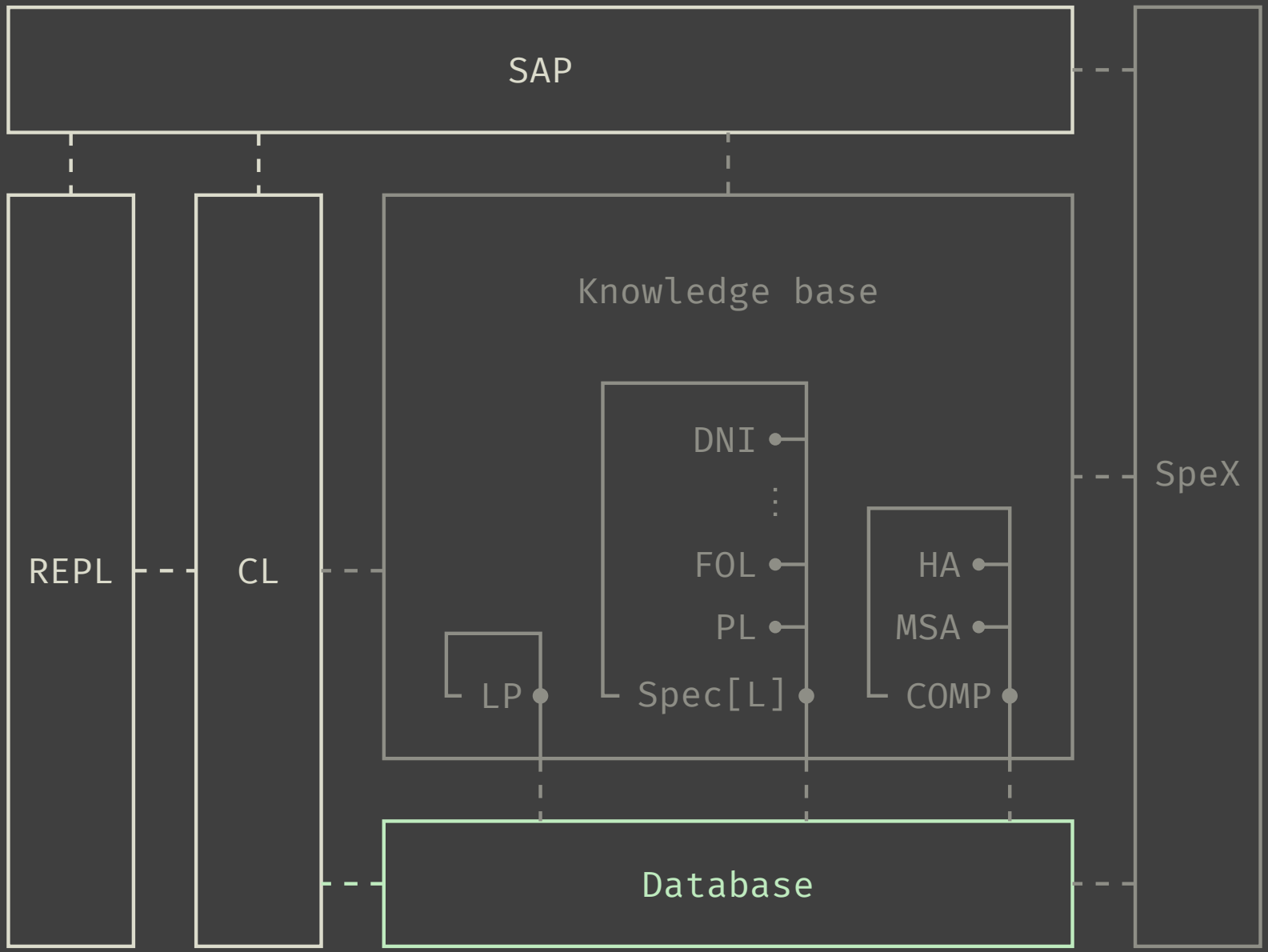
An overview of SpeX (overly simplified)



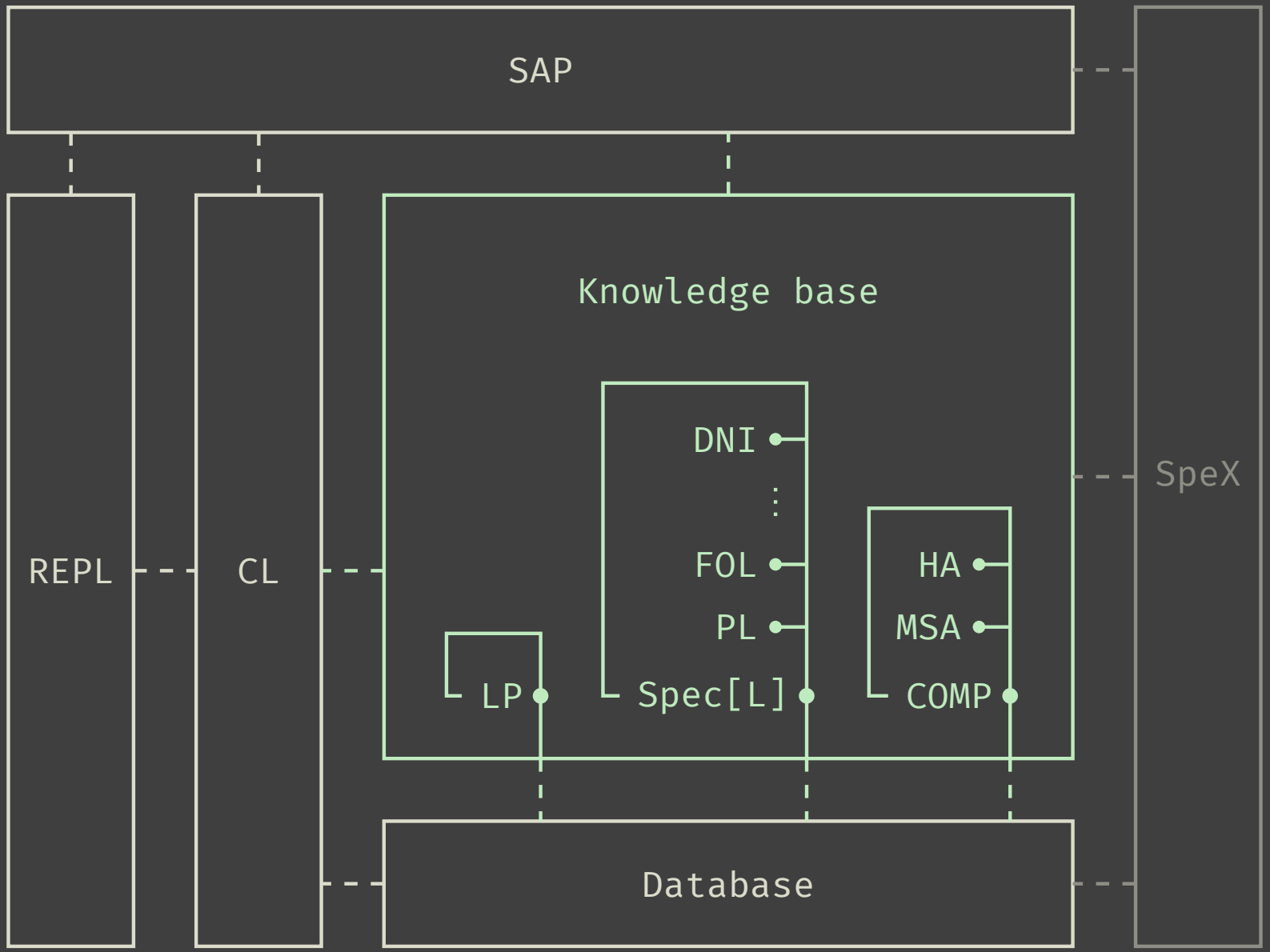
An overview of SpeX (overly simplified)



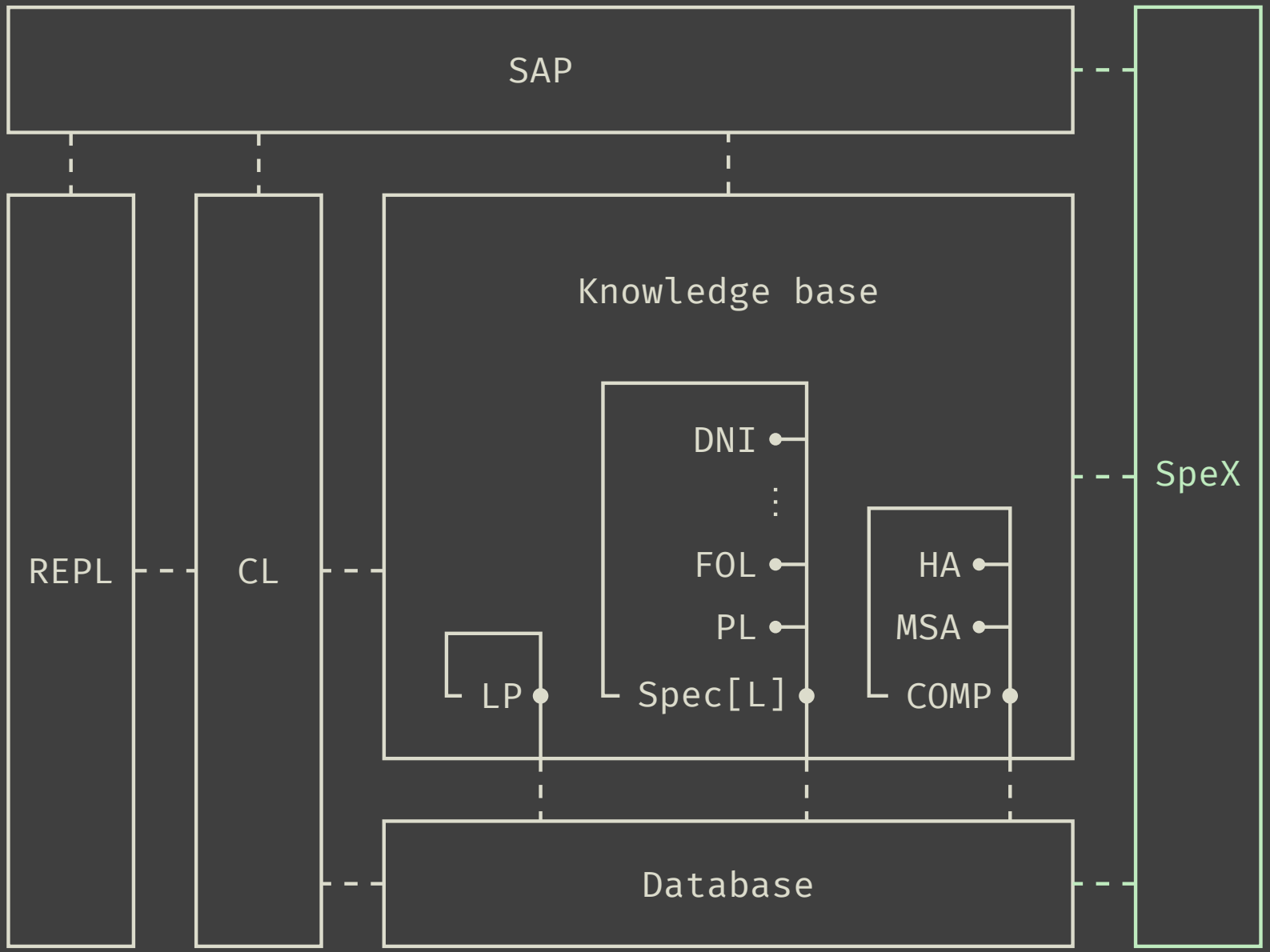
An overview of SpeX (overly simplified)



An overview of SpeX (overly simplified)



An overview of SpeX (overly simplified)



To understand how it works ...

* we take a look at REPL interpreters
  - objects of class INT
  - interact with REPL objects (streams)

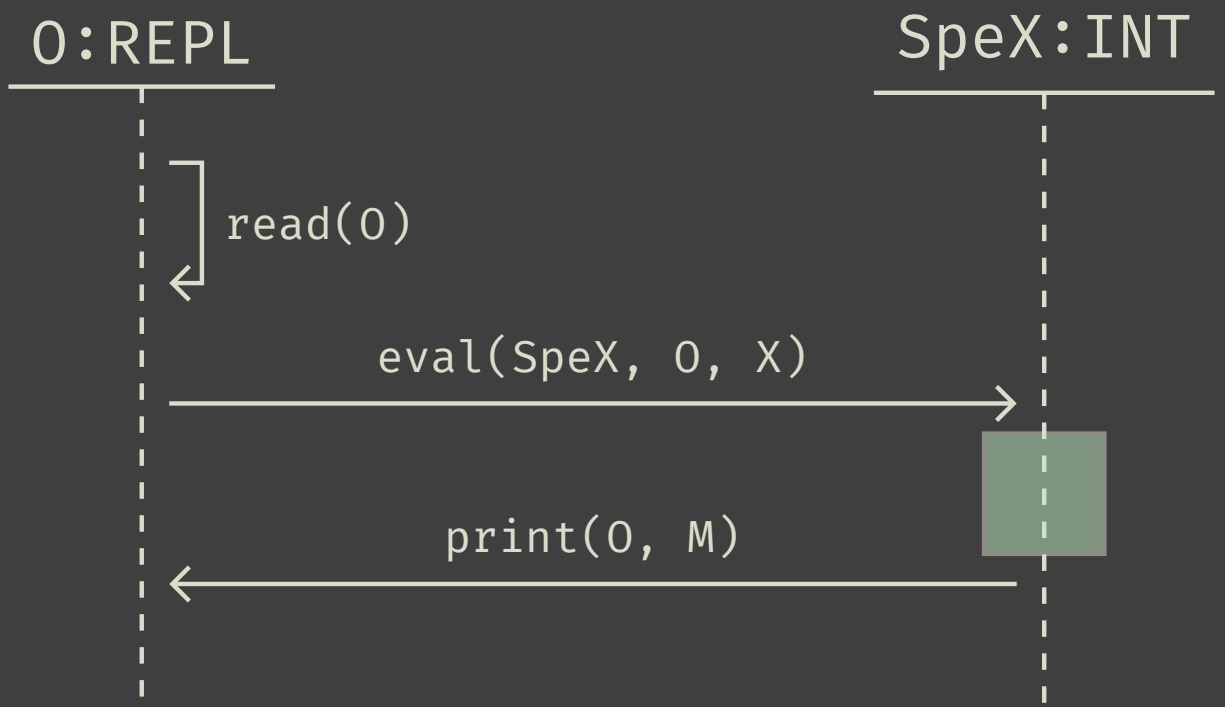
* they receive messages of the form

```
eval(I, O, X)
```

* reply with messages of the form

```
print(O, Text)
warn(O, Text, Arguments)
read-more(O) ...
```

A basic execution scenario



Integrating new languages into SpeX

* by means of processors
  - objects of class PROC
  - interact with SpeX (the object)

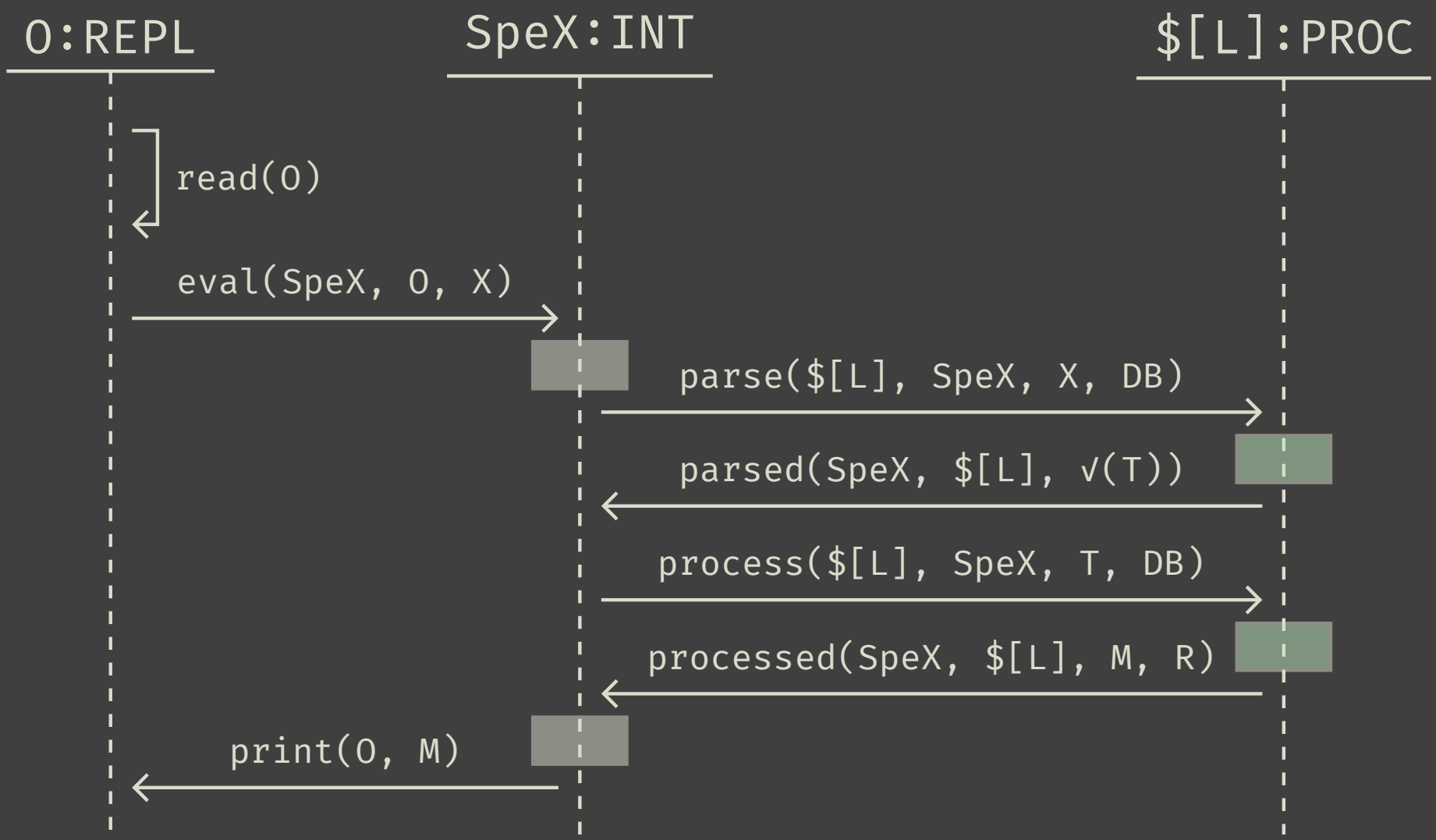
* receive messages of the form

```
parse($[L], SpeX, Input, DB)
process($[L], SpeX, AnnotatedTerm, DB)
```

* reply with messages of the form

```
parsed(SpeX, $[L], ParsingOutcome)
processed(SpeX, $[L], Text, Record)
```

A basic execution scenario (cont.)



Example: Spec[DNI]

```
spec Bind is
  including Base .
  mod __bind_ : Protein Organelle >< Coat .
  …
  sen store k:Nominal
      forall-local {p:Protein, o:Organelle}
      [ p o bind z:Coat ]
      (forall-local {o':Organelle}
        brane(o') = @(k) brane(o))
      and
      (forall-local {c':Coat}
        c' = z implies brane(c') = @(k) brane(o))
      [label: bind-effect] .
endspec
```

Example: Spec[DNI]

```
spec Bind is
  including Base .
  mod __bind_ : Protein Organelle >< Coat .
  ...
  sen store k:Nominal
      forall-local {p:Protein, o:Organelle}
      [ p o bind z:Coat ]
      (forall-local {o':Organelle}
        brane(o') = @(k) brane(o))
      and
      (forall-local {c':Coat}
        c' = z implies brane(c') = @(k) brane(o))
      [label: bind-effect] .
endspec
```

Example: Spec[DNI]

```
spec Bind is
  including Base .
  mod __bind_ : Protein Organelle >< Coat .
  ...
  sen store k:Nominal
      forall-local {p:Protein, o:Organelle}
      [ p o bind z:Coat ]
      (forall-local {o':Organelle}
        brane(o') = @(k) brane(o))
      and
      (forall-local {c':Coat}
        c' = z implies brane(c') = @(k) brane(o))
      [label: bind-effect] .
endspec
```

Example: Spec[DNI]

```
spec Bind is
  including Base .
  mod __bind_ : Protein Organelle >< Coat .
  ...
  sen store k:Nominal
      forall-local {p:Protein, o:Organelle}
      [ p o bind z:Coat ]
      (forall-local {o':Organelle}
        brane(o') = @(k) brane(o))
      and
      (forall-local {c':Coat}
        c' = z implies brane(c') = @(k) brane(o))
      [label: bind-effect] .
endspec
```

Example: COMP

```
bobj WATCH is
  syncing (UP-TO-24-COUNTER as HOUR)
      and (UP-TO-60-COUNTER as MINUTE)
      and (UP-TO-60-COUNTER as SECOND) .
  op _:_:_ : Nat Nat Nat → State .
  act tick_ : State → State .
  act inc-min_ : State → State .
  ...
endbo
```

```
open WATCH
  check tick inc-min (H:Nat : M:Nat : S:Nat)
      ~ inc-min tick (H:Nat : M:Nat : S:Nat)
  forall M:Nat < 60 = true
     and S:Nat < 60 = true .
close
```

Example: COMP

```
bobj WATCH is
  syncing (UP-TO-24-COUNTER as HOUR)
      and (UP-TO-60-COUNTER as MINUTE)
      and (UP-TO-60-COUNTER as SECOND) .
  op _:_:_ : Nat Nat Nat → State .
  act tick_ : State → State .
  act inc-min_ : State → State .
  ...
endbo
```

```
open WATCH
  check tick inc-min (H:Nat : M:Nat : S:Nat)
      ~ inc-min tick (H:Nat : M:Nat : S:Nat)
  forall M:Nat < 60 = true
     and S:Nat < 60 = true .
close
```

Obtaining SpeX

* from the GitLab repository:

  https://gitlab.com/ittutu/spex

* then, if Maude 3(.2) is installed:

```
./configure
make
[sudo] make install
```

Happy hacking!